Software Engineering Institute

Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009)

Grace A. Lewis
Dennis B. Smith
Ned Chapin
Kostas Kontogiannis

February 2010

SPECIAL REPORT
CMU/SEI-2010-SR-004

Research, Technology, and System Solutions Program

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

List of Tables

Acknowledgments

There are many people that we would like to thank for making this workshop a success.

First of all, we would like to thank the authors and presenters that shared their work with us.

- Sedigheh Khoshnevis, Pooyan Jamshidi, Ali Nikravesh, Alireza Khoshkbarforoushha, Reza Teimourzadegan and Fereidoun Shams (Shahid Beheshti University, Iran)
- Hausi A. Müller, Priyanka Gupta, Ron Desmarais, Alexey Rudkovskiy, Norha Milena Villegas and Qin Zhu (University of Victoria, Canada)
- Leho Nigul (IBM Center for Advanced Studies, Canada)
- Quyen L. Nguyen (National Archives and Records Administration, USA)
- Paulo Rupino da Cunha, Paulo Melo and Catarina Ferreira da Silva (University of Coimbra, Portugal)

Next we would like to thank the workshop attendees that contributed to the lively discussion.

- Maria Teresa Baldassarre (University of Bari, Italy)
- Priyanka Gupta (University of Victoria, Canada)
- Pooyan Jamshidi (Shahid Beheshti University, Iran)
- Panos Linos (Butler University, USA)
- Hausi Müller (University of Victoria, Canada)
- Norha Villegas (University of Victoria, Canada)
- Chunchal Roy (University of Saskatchewan, Canada)
- Paulo Rupino (University of Coimbra, Portugal)
- Mike Smit (University of Alberta, Canada)
- Eleni Stroulia (University of Alberta, Canada)
- Scott Tilley (Florida Institute of Technology, USA)
- Joris Van Geet (University of Antwerp, Belgium)
- Ken Wong (University of Alberta, Canada)
- Carl Worms (Credit Suisse, Switzerland)
- Namhoo Yoo (DoS/HA Contractor, USA)
- Hamzeh Zawawy (University of Waterloo, Canada)

We thank our program committee, who made sure we had a set of high quality papers to frame our workshop.

- Massimiliano Di Penta (University of Sannio, Italy)
- Marin Litoiu (York University, Canada)
- Liam O'Brien (National ICT, Australia)
- Mira Kajko-Mattsson (Stockholm University and Royal Institute of Technology, Sweden)

- Hausi Müller (University of Victoria, Canada)
- Harry Sneed (ANECON GmbH, Austria)
- Scott Tilley (Florida Institute of Technology, USA)

Finally, we thank the ICSM 2009 organizers for all their help, especially Melanie Calvert, Eleni Stroulia and Ken Wong from the University of Alberta.

Abstract

The 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009) was held at the 25th International Conference on Software Maintenance (ICSM 2009) in Edmonton, British Columbia, Canada on September 21, 2009.

The goal for MESOA 2009 was to discuss current efforts in the maintenance and evolution of service-oriented systems and identify research challenges in addressing existing gaps and problems. The workshop had 20 attendees, representing industry and academia. Papers and presentations contributed original work, and the discussions highlighted additional work that is being done as well as new research challenges. Presentations from the workshop can be found at http://www.sei.cmu.edu/workshops/mesoa/2009/.

This report includes selected papers presented at the workshop that highlight important issues within the established SOA research agenda.

1 Introduction

The 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems was held at the 25th International Conference on Software Maintenance (ICSM 2009) on September 21, 2009.

There are many successful case studies of service-oriented architecture (SOA) adoption, mainly in commercial enterprises, where the main goal for SOA implementation is internal integration and business process improvement. Despite recent reports that "SOA is Dead"[1], the reality is that SOA is currently the best option available for systems integration and leverage of legacy systems. According to a 2007 Gartner Group report, 50% of new mission-critical operational applications and business processes were designed in 2007 around SOA, and that number will be more than 80% by 2010. The technologies to implement SOA will most probably change over time, but the concepts will remain valid. From a maintenance and evolution perspective, there are two concerns: (1) deployed service-oriented systems will have to be maintained and evolved, and (2) legacy systems will migrate to SOA environments to make their legacy functionality available to other systems and applications. The main goal of this workshop was to create a focal point and an ongoing forum for researchers and practitioners to share results and open issues in the area of maintenance and evolution of service-oriented systems.

In 2007, the Software Engineering Institute started assembling a SOA Research Agenda based on a proposed life cycle for service-oriented systems that emphasizes the relationship between business strategy and SOA strategy. The core of the agenda is a taxonomy of research topics where new, more, or different research is needed to support short-term and long-term strategic SOA adoption. The structure of each topic is rationale, current efforts, and a set of challenges and gaps. Since then, multiple workshops have been organized to validate, discuss and evolve the agenda, as well as specific topics within the agenda. One of these workshops (now in its third instance) is the Workshop on Maintenance and Evolution of Service-Oriented Systems (MESOA) that focuses specifically on the topic of maintenance and evolution within the Engineering area of the SOA research agenda.

During MESOA 2007, topics such as service identification, concept location, and service testing were presented as techniques to support maintenance and evolution of service-oriented systems. In addition, the effect of business value and autonomic computing on maintenance and evolution of service-oriented systems were discussed. Specific research challenges that were identified included:

- Balance of maintenance types in service-oriented systems
- Characterization of the preparation phase for SOA adoption
- Effects of governance and compliance on maintenance and evolution of service-oriented systems
- Effects of process maturity on service-oriented systems maintenance

[1] http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services/comments/page/2/

- Implications of autonomic computing and monitoring for maintenance and evolution of service-oriented systems
- Models for return on investment (ROI) of SOA adoption and evolution
- Retirement of services
- Runtime validation of compliance to evolving business processes
- Service identification from legacy code
- Software reusability effects on SOA maintenance
- Testing of services and service-oriented systems
- Trust, certification and verification of services

In MESOA 2008, the focus was on research and advances in specific areas that were identified by the SOA Research Agenda as critical for maintenance and evolution in a dynamic, heterogeneous and potentially distributed development and maintenance environment, such as testing, legacy system migration and runtime monitoring. Specific research challenges that were identified included

- Testing in SOA environments
  - exception-oriented testing
  - effects of SOA 2.0 and event-driven architecture on testing
  - effects of dynamic service composition on testing
  - test automation that generates large volumes of relevant test cases
  - service granularity appropriate for testing
  - rules for web service interface definition
  - metrics for measuring web service interface definition quality—static and runtime
- Legacy system migration to SOA environments
  - When is new development a better option than legacy system migration or simple web migration?
  - Is it possible to define cost-effective SOA migration strategies for non-decomposable legacy systems?
  - Metrics for system decomposability
  - Mapping of business objectives to migration strategies
- Runtime monitoring of service-oriented systems
  - Construction of models for indicators
  - Managing and leveraging uncertainty
  - Making control loops explicit in service-oriented systems
  - Maintainability concerns for self-adaptive SOA systems compared to static, non-adaptive SOA systems

The focus for MESOA 2009 was to continue sharing current efforts in the maintenance and evolution of service-oriented systems and identify areas of future work to address existing gaps and problems, present a set of longer term research challenges in maintenance and evolution of ser-

vice-oriented systems, and understand the maintenance and evolution challenges of deployed service-oriented systems.

The workshop started with an introduction of the SOA research agenda, some of the challenges defined in the agenda under maintenance and evolution, and a proposed set of maintenance characteristics. The day was then divided into five sessions:

- Session 1: Tools for Migration to SOA Environments
- Session 2: Case Studies in Systems Migration to SOA Environments
- Session 3: SOA Governance and Service-Oriented Systems Evolution
- Session 4: Longer Term Research Topics in Maintenance and Evolution of Service-Oriented Systems
- Session 5: Panel - Challenges for Maintenance and Evolution of Deployed Service-Oriented Systems

Each session included one or more presentations, plus a guided discussion, with the goal of identifying remaining research challenges in each area. At the end, the results of the workshop were summarized and next steps were presented.

Section 2 provides a brief summary of the SOA research agenda, with a particular focus on challenges for maintenance and evolution of service-oriented systems. Section 3 is a paper that characterizes SOA implementation projects as mainly maintenance projects, along with the challenges that this presents. Section 4 contains a paper that presents an automated approach for service model evolution, along with a preliminary implementation of a tool to support the approach. Section 5 is a case study in the Portuguese telecom industry and Section 6 is a case study in the U.S. National Archives and Records Administration. Section 7 is a paper on how SOA governance optimizes the business and evolution of SOA systems. Section 8 is a summary of the workshop that contains details of the discussions that took place, along with an overview of the presentations (including the panel on challenges for the maintenance and evolution of deployed service-oriented systems), as well as conclusions and next steps.

2 Overview of a Research Agenda for Service-Oriented Architecture: Maintenance and Evolution of Service-Oriented Systems

The SEI started developing a SOA Research Agenda in 2007, which was based on a proposed life cycle for service-oriented systems that emphasizes the relationship between business strategy and SOA strategy. The core of the agenda is a taxonomy of research topics where new, more, or different research is needed to support short-term and long-term strategic SOA adoption. This section provides a brief overview of the SOA research agenda and identifies research needs for maintenance and evolution of service-oriented systems. For more details on the research agenda, see the technical reports by Lewis, et al (Kontogiannis, Lewis and Smith 2007, Lewis, Kontogiannis and Smith 2010).

2.1 Overview of the SOA Research Framework

The SOA research framework includes:

- A proposed service-oriented systems development life cycle
- A taxonomy of SOA research areas related to engineering, business, operations and cross-cutting concerns

These are summarized at a high level in the following sub-sections. For more details, see the SEI technical report by Lewis, et al (Lewis, Kontogiannis and Smith, A Research Agenda for Service-Oriented Architectures 2010).

2.1.1 SOA Solution Space and Problem Space

In a service-orientation adoption setting, an organization develops a service strategy that takes into account the organization’s business drivers, context, and application domain. In order to accomplish the service strategy, the organization must generate plans to achieve the goals and objectives outlined by the strategy. The execution of these plans requires business, engineering, and operations decisions to be made while simultaneously taking into consideration cross-cutting concerns, such as governance, security, risk management, social and legal issues, and training and education. The development of the SOA strategy and SOA plans take into consideration and affect the organization's business model and service model. Figure 1 illustrates the SOA problem space and solution space at a high level.

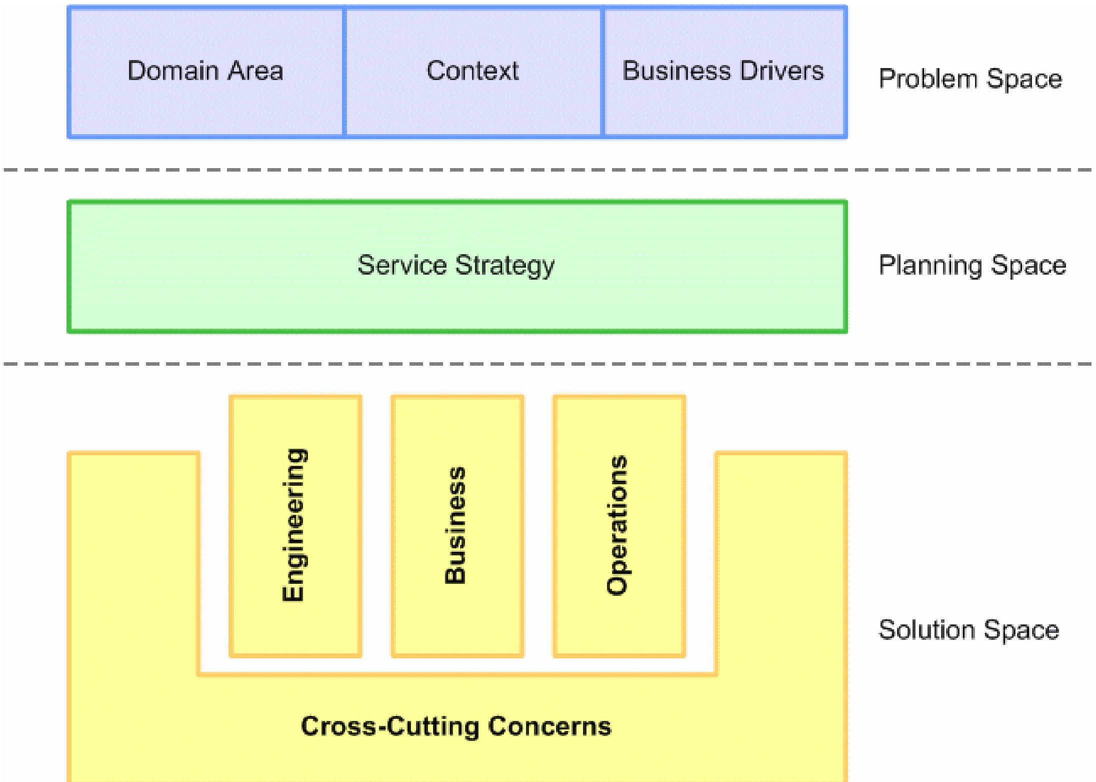

Figure 1 Overview of the SOA Problem Space and Solution Space

2.1.2 Service-Oriented Systems Development Life Cycle

The proposed service-oriented systems development life cycle recognizes that SOA adoption requires an iterative approach to systems development that reflects the strong link between business strategy and development strategy. SOA adoption is not “all or nothing”—one of the benefits of SOA adoption is that systems and system components can be deployed gradually. The main differences between this life cycle and other iterative development frameworks, such as the IBM Rational Unified Process (RUP), are the emphasis on activities to establish and analyze the relationship with business goals at the beginning of the cycle, the emphasis on evaluation at the end of the cycle, and the specification and review of business objectives at the end of the cycle so that the requirements for each iteration follow business objectives.

Each iteration is composed of a set of phases, activities and indicators. Table 1 Illustrates a notional mapping between phases, activities and indicators. The cells simply indicate that a greater or lesser amount of an activity takes place (or an indicator is required) during a specific phase. For example, the activity in the Strategic Analysis Phase (P1) primarily uses activities A1, A2, and A3 (Business Context Understanding, Business Objectives Specification, and Risk Analysis).

Table 1 Mapping between Phases, Activities and Indicators

PHASES	P1: Strategic Analysis	P2: Planning	P3: Construction	P4: Transition	P5: Production
ACTIVITIES					
A1: Business Context Understanding	+++				++[1]
A2: Business Objectives Specification	+++	++	+		
A3: Risk Analysis and Initial Requirements Gathering	+++	+++	++	+	
A4: Prototyping and Requirements Tuning		++	+		
A5: Design and Implementation		+	+++[2]	++	
A6: Integration and Testing			++	+++	++
A7: Deployment				+++	++
A8: Maintenance			+	+	+++
A9: Management				++	+++
INDICATORS					
E1: Financial Indicator Measurements	++	+++			
E2: Technology Indicator Measurements	+	++	++	+	++
E3: User Rating Measurements				++	+++
E4: Compliance Indicators	+	++	+	++	+++

1 Indicates a state of “perpetual beta”—requirements are not fixed; requirements for each iteration follow business objectives.

2 Implementation, Integration and Adoption, and Maintenance and Management are followed according to the development process used (RUP, Agile, etc.).

2.1.3 Taxonomy of SOA Research Areas

The development of a service-oriented system requires business, engineering, and operations decisions to be made, as well as other cross-cutting decisions. The taxonomy of research topics, shown in Figure 2, is divided into these decision areas. The research topics correspond to areas where new, different, or additional research is needed to support a strategic approach to service-oriented systems development.

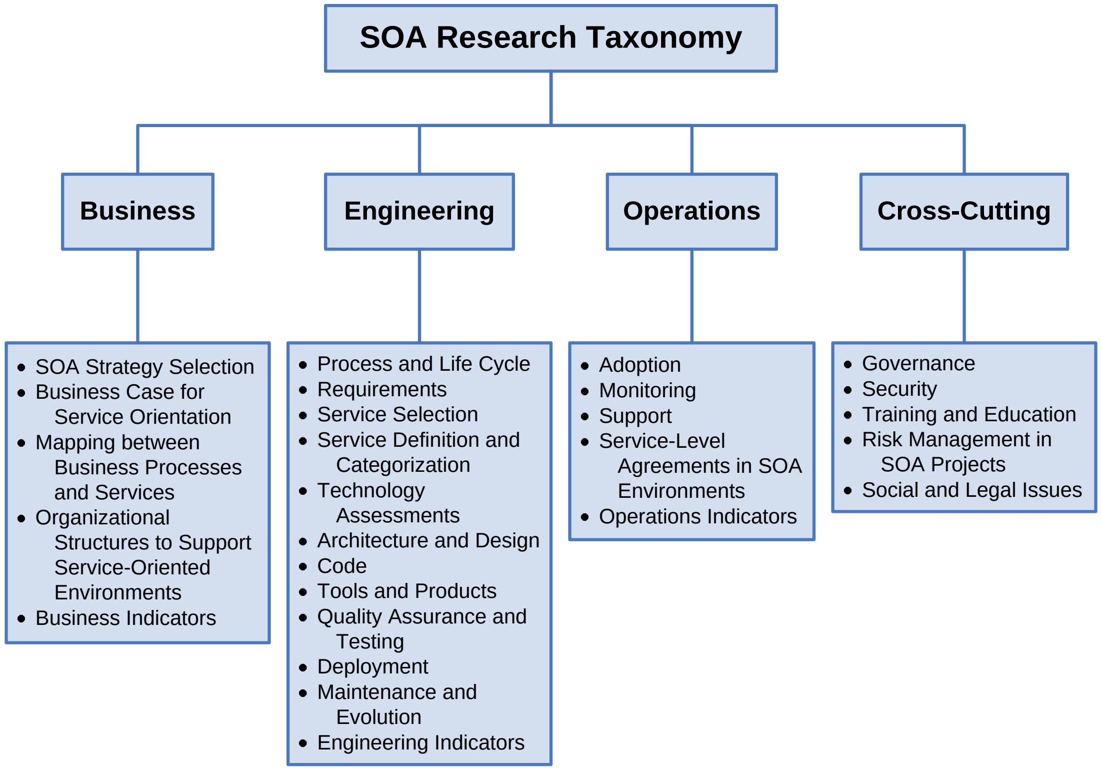

Figure 2 SOA Research Taxonomy

Each research topic is analyzed in terms of: 1) the rationale of why it is an important research topic, 2) current efforts in the area, and 3) challenges and gaps in the area. The goal for the research agenda is to identify topics for industrial or academic research that can make a difference for strategic SOA adoption.

2.1.4 Maintenance and Evolution Research Areas

Service-oriented systems are significantly different from traditional systems, resulting in new research issues that need to be addressed. These differences include: 1) the diversity of service consumers and service providers, 2) shorter release cycles because of the capability of rapidly adapting to changing business needs, and 3) the potential to leverage legacy investments with potentially minimal change to existing systems.

What does maintenance and evolution look like in this dynamic, heterogeneous and potentially distributed development and maintenance environment? Ongoing and projected research in the topics presented in Figure 3 and summarized below will provide a focus for finding answers to this question. For more details on challenges and concerns for maintenance and evolution of service-oriented systems, see the technical reports by Lewis, et al (Kontogiannis, Lewis and Smith 2007, Kontogiannis, Lewis and Litoui, et al. 2007, Kontogiannis, Lewis and Smith 2008, Lewis, Kontogiannis and Smith 2010).

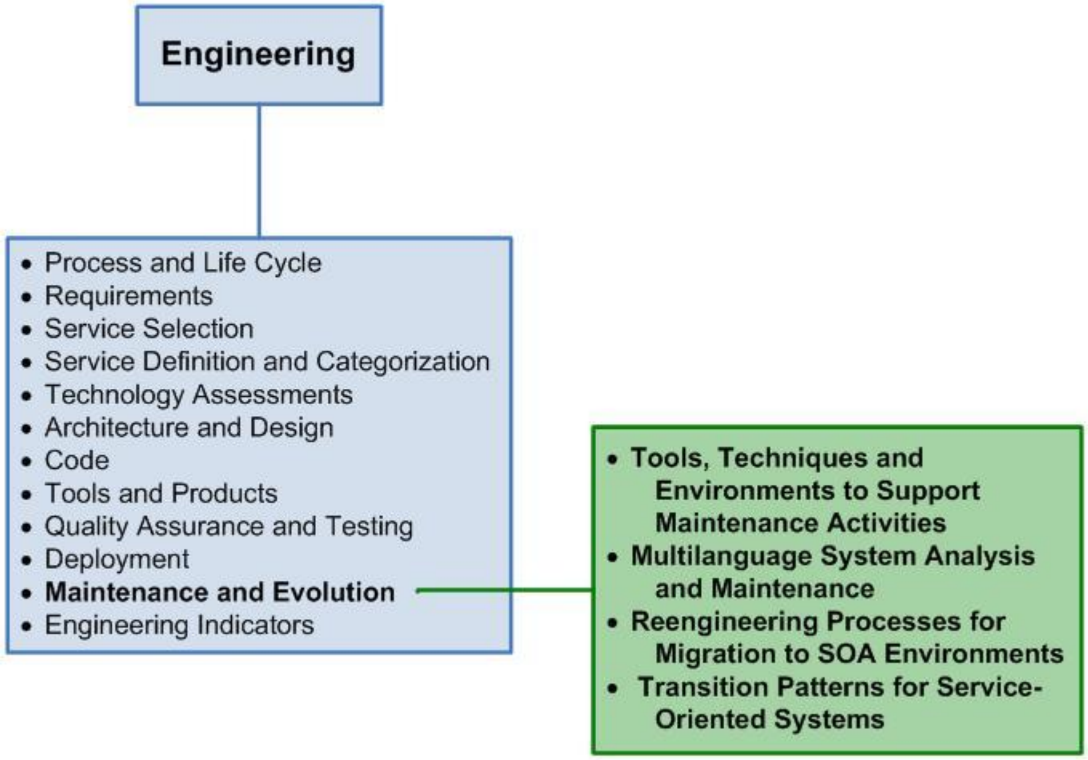

Figure 3 Research Topics for Maintenance and Evolution of Service-Oriented Systems

Tools, Techniques and Environments to Support Maintenance Activities

The complexity of the maintenance process in a SOA environment is greater than that in a traditional system, especially if there are external consumers and providers involved. The development of specialized methods and tools to support the maintenance and evolution of large service-oriented systems is in the early stages. Current efforts seem to indicate that maintenance activities for service-oriented systems are not that different than in traditional systems. However, we are still in the stage where most service-oriented systems are deployed for internal integration, where there is still some control over deployed services.

We believe that the emergence of the market for third-party services and the deployment of more service-oriented systems that cross organizational boundaries will require changes to current maintenance practices. From an engineering perspective, processes to support the incremental evolution of service-oriented systems, configuration management, impact analysis and versioning in this environment are challenges. From a business perspective, the organizational structures and roles to support maintenance of service-oriented systems as well as models to support the development and maintenance of shared services are areas of much needed research.

Changes that do not require modifications to the service interface will potentially have no impact on service consumers. However, a change in technology may have a negative effect on provided quality of service, even if the interface remains the same. Changes that do require modifications to the service interface can have a potentially large impact on service consumers. Important research issues are related to maintenance of multiple interfaces, impact analysis techniques for service consumers and providers in this environment, change notification mechanisms for service consumers, and proper use of extensibility mechanisms in messaging technologies.

Service-oriented systems can be deployed over a wide geographic area and on a set of different server computers. Owners of the service-oriented system may not have control over some of the services used. Despite the fact that robust techniques for configuration management in centralized systems are available, there are open issues with respect to managing change in distributed code bases and code repositories, especially when third-party services are involved. Furthermore, there may be additional requirements for the configuration management of large service-oriented systems. As a result, the development of a unified model for managing and controlling change in such systems remains an open research issue.

Multilanguage System Analysis and Maintenance

One of the benefits associated with SOA, and especially with web services, is true platform independence. Even though standard interfaces are exposed, the underlying service implementation could be written in almost any language. While this is a huge benefit, it makes looking at the system as a whole difficult. Most research in this area is limited to small size projects and a small number of languages. This is a problem for an environment that promotes platform independence. In the case of third-party service providers, access to source code is usually not possible. In these cases, an important area of research is the identification of the type of information that service providers would need to expose to service consumers—in interfaces, or service registries or repositories—that wish to do code analysis, as well as tools and techniques to support the process.

Reengineering Processes for Migration to SOA Environments

Because it has characteristics of loose coupling, published interfaces, and a standard communication model, SOA enables existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems. Migration of legacy assets to services has been achieved within a number of domains, including banking, electronic payment, and development tools, showing that the promise is beginning to be fulfilled. While migration can have significant value, any specific migration requires a concrete analysis of the feasibility, risk and cost involved. The strategic identification and extraction of services from legacy code is also crucial.

The ideal reengineering process would be one that implements the full SOA-Migration Horseshoe model (Winter and Ziemann 2007) that starts from implemented legacy code, develops legacy software and legacy enterprise models, and then transforms the models to services and components. Currently, there are techniques and tools that implement portions of the horseshoe, but not the full horseshoe. An important area of research would be the development of concrete processes that implement the horseshoe, as well as tools (or suites of tools) to support the process. The automation of this process would be a complex task, but one that is worth investigating.

In addition, a challenge is mining legacy code for services that have business value. Research topics in this area include

- Tools and techniques for analyzing large source code bases to discover code that is of business value
- Metrics for "wrapability" and business value to determine reusability (Sneed 2007)

- Application of feature extraction techniques to service identification, given that services usually correspond to features (Sneed 2007)

Transition Patterns for Service-Oriented Systems

SOA adoption can enable incremental system modernization. For example, legacy system components can be initially wrapped using Web Services technology and replaced with newer components incrementally. As long as the interfaces remain stable, service consumers only have to be modified once to initially access the new services. However, this means that throughout the life cycle of the project, there will be a mix of migrated legacy components, legacy components waiting to be migrated, and legacy components that will not be migrated. Legacy components include application front ends, business logic, data logic, and actual data. A major challenge for incremental migration is therefore the minimization of "throw-away" cost and effort to support intermediate system states. A significant need exists for research that addresses the total cost of a SOA effort that includes development, implementation, data migration, and incremental updates. Such a model would also provide one of the inputs for realistic ROI calculations.

References

Kontogiannis, Kostas, et al. "The Landscape of Service-Oriented Systems: A Research Perspective." *Proceedings of the International Workshop on Systems Development in SOA Environments.* Minneapolis: International Conference on Software Engineering (ICSE) 2007, 2007.

Kontogiannis, Kostas, Grace Lewis, and Dennis Smith. "A Research Agenda for Service-Oriented Architecture." *Proceedings of the Second International Workshop on Systems Development in SOA Environments.* Leipzig: Association for Computing Machinery, 2008.

—. "The Landscape of Service-Oriented Systems: A Research Perspective for Maintenance and Reengineering." *Proceedings of the 11th International Workshop on Service-Oriented Architecture Maintenance and Reengineering (SOAM).* Amsterdam, The Netherlands, 2007.

Lewis, Grace, Kostas Kontogiannis, and Dennis B. Smith. *A Research Agenda for Service-Oriented Architectures.* CMU/SEI 2010-TR-003, Pittsburgh: Carnegie Mellon University, 2010.

Sneed, H. "Migrating to Web Services: A Research Framework." *Proceedings of the International Workshop on SOA Maintenance Evolution.* Amsterdam: Association for Computing Machinery, 2007.

Winter, A. and Ziemann, J. "Model-Based Migration to Service-Oriented Architectures." *Proceedings of the International Workshop on SOA Maintenance Evolution.* Amsterdam: IEEE Computer Society, 2007.

3 Research on Maintenance Characteristics of SOA Systems

Ned Chapin (InfoSci Inc., USA)

Abstract

Sufficient experience with implementing service-oriented architecture (SOA) has been acquired so that the maintenance characteristics of SOA implementations can be evaluated. The International Standards Organization and the IEEE have concurred on a software maintenance standard that is appropriate for evaluating SOA implementations' software maintenance characteristics. The evaluation reported here considers four SOA implementation situations: pilot implementations, both with and without wrappers and middleware such as an enterprise service bus (ESB), and subsequent implementations, both with and without wrappers and middleware, such as an enterprise service bus. The evaluation finds that maintenance characteristics dominate in SOA implementations. This offers at least ten challenging opportunities for formal research on SOA implementations.

3.1 Objective

In describing the benefits obtainable from implementing SOA systems, the published literature very commonly has listed reduction of the software maintenance burden(Josuttis 2007, Oracle 2005). The literature also often has listed improving the capability to be more agile or more flexible and rapidly responsive in meeting dynamic changes in the SOA using organization's environment, and reduced IT costs (Freivald 2008, Mahmood 2007). Now that more than half a decade of SOA experience has been accumulated, the benefits from using SOA can now be more thoroughly evaluated. This paper offers an evaluative status report after reviewing maintenance characteristics and SOA characteristics. The report may help to identify some potentially fruitful research opportunities to clarify the maintenance characteristics of SOA-implemented systems.

3.2 Maintenance Characteristics

The characteristics of software maintenance have been extensively reported in the published literature. Many of these have been summarized in the ISO/IEC 14764 international standard on maintenance, and the IEEE Std 14764-2006 that in 2006 replaced the older IEEE Std 1219, extending the revised standard to apply both to software and to systems incorporating software (International Organization for Standardization 2006).

Like other software standards, the 14764 standard is process oriented, but stands separate from the standards about software development. Both the maintenance and development standards mostly ignore some common recent increasing complexities in information system situations. Mashups are one example of this, where the software components incorporated may be from and/or owned by diverse independent sources.

Consistent with the 14764 international/IEEE standard, the characteristics of maintenance that appear to be most common and relevant for this evaluation are these:

- **Existing software.** Maintenance is a software modification process done on existing software and/or system anytime after its initial delivery, but includes any planning done prior to initial delivery about the processes done after initial delivery, including retirement.
- **Components.** Maintenance processes are comprised of many component processes depending upon the type of software maintenance and upon the characteristics of the software and its artifacts (software products), and the characteristics of the operation and use of the software.
- **Software maintenance life cycle.** Major component processes may include software maintenance life-cycle processes such as planning, analyzing, designing, programming, testing, reviewing, migrating, and retiring, and their many more-specific components.
- **Artifacts.** Common maintenance process artifacts that may be modified during maintenance include modification requests, plans, documentation, testing materials, source code, reports, etc.
- **All types.** Maintenance types have historically and in the standard been based upon the intentions of the requester of the maintenance. The standard includes all four of those types: corrective, perfective, adaptive, and preventive. An objective evidence basis for maintenance types can give a richer, more useful, and more specific classification for the types of maintenance as actually done, but is not part of the 14764 standard currently.

The 14764 standard does not directly define “initial delivery,” but instead in its § 3.11 uses the military terminology of “transition” instead of the more common terminology of “handover” or “turnover.” Planning that handover or transition is included in the 14764 standard. Under either name, that process marks the nominal end of software development and the nominal start of software maintenance. In practice, the usual criteria that software needs to satisfy to be eligible for handover has two parts. The first criterion is that the software is deemed ready for use, and the second criterion is that the software has successfully passed unit, integration, and acceptance tests, or that it satisfies the applicable quality assurance requirements. Software satisfying the nominal criteria can be deployed (released for routine production use). In practice, the initial use of the software in production may expose some oversights or shortfalls that lead to requests to begin some type of software maintenance.

The 14764 standard does not include, incorporate, or recognize the common practice of classifying software work on the basis of person-hours, budget amount, or budget category. For example, some business and governmental organizations automatically classify all software work requiring more than two person-months of personnel time as being software development. Additionally, some organizations put a cap on the maintenance part (such as 20%) out of the total amount of software work that they will permit or fund during any given budgeting time period, such as three fiscal years. Any software work beyond the cap is regarded (and treated) as being software development.

The 14764 standard does not specify what or who does the maintenance. The implication from the standard is that maintenance is accomplished as a set of collaborative activities involving people and the use of computers. The source of the people is not limited to any one source—the people

may be using the organization's own employees, or contract personnel, or interns, or personnel at vendor or contractor facilities, or pro bono personnel, etc., in any combination. The sources of the computers are also not limited.

The 14764 standard distinguishes maintenance from development and other kinds of processes involving computer software. The International Standards Organization (ISO) and the IEEE have other standards covering development and many other kinds of processes, but no standard is yet present for SOA.

3.3 SOA Characteristics

Since SOA is relatively young, its characteristics are less uniformly recognized and adhered to than if it were more mature. SOA implementations conform to the classic computer application model, but focus on reusable functions (such as "find inventory location") that may support or be a service to other functions. The inputs of SOA components are primarily data arising from the using organization's interactions with that organization's internal components and its environment, and from files and databases of such data. SOA components' outputs are primarily data for the using organization's current use in interactions with that organization's internal components and its environment, or stored in files and databases for later use.

The classic SOA implementation has most of its components implemented as web services using XML (extended markup language) as the format for inter-component communications, such as data flows (Ferris and Farrell 2001, Ma 2005, Willis 2008). Also, SOA implementations are almost always made by doing software projects (Lewis, Morris, et al. 2007). From the abundant SOA literature that has been fielded thus far, the following SOA characteristics appear to be among the most common and relevant for this evaluation:

- **Project.** To make a SOA implementation at a computer-using organization, the usual practice is to introduce SOA through a focused pilot project. When that initial project has been successfully implemented, the usual practice is to extend the scope of the SOA implementation by doing a series of interacting subsequent SOA projects (Knorr and Rist 2005).
- **Bounds.** The focus of a SOA project or set of successive SOA projects is usually a set of the SOA-using organization's existing data handling processes. The selected set is usually bounded to reduce requirements creep. These processes can be defined and documented by any of several means such as by business process modeling (BPM), and the process boundaries defined in terms of data inflows and outflows (Chapin 2008, Geminiue 2008). The inflows of data are either data coming from other processes or data stores within the organization, or data coming into the organization from other organizations, such as from a client, for example. The outflows of data are either data going to other processes or to data stores within the organization, or data going from the organization to other organizations, such as to a regulatory agency, for example. Files and databases are examples of data stores.
- **Replace.** The constituent components in a SOA implementation can be or may be replacements for one or more existing interacting software components in the SOA-using organization, and result in the retirement of those existing components (Reddy, et al. 2009).
- **Wrapper.** If a SOA implementation does not replace one or more interacting software components in the SOA-using organization, then it may incorporate those software components

into the SOA implementation by wrapping them or by providing middleware (such as an ESB), or by some other combination (Manes 2007).

- **Distributed.** A SOA implementation can use its Web services components or other components to access the Internet for communication with the using organization's owned geographically distributed hardware or software or data, or some combination of these. The typical SOA implementation operates in a distributed manner, incorporating the SOA-using-organization's hardware and software and data that are located remotely, as well as its hardware, software, and data that are located locally (Schepers, Iacob and van Eck 2008).
- **Resource.** A SOA implementation can use its Web services components or other components to access the Internet in order to incorporate non-owned hardware or software or data resources into the SOA implementation, wherever they may be located, such as in the cloud or from specific resource suppliers, potentially through software-as-a-service (SaaS) (Nitu 2009, Sedayao 2008).

In practice, an organization's initial SOA implementation almost never attempts to cover an entire enterprise or organization (Baer 2007). Instead, the common practice is a phased series of integrated SOA software projects starting with one small initial or pilot project, where the entire series of projects does not even attempt to cover the entire enterprise or organization. The consequence is that currently (and for at least the coming decade) nearly every organization's accomplished SOA implementation will be a partial implementation, interoperating with the organization's remaining non-SOA still-operational software.

In order to streamline the work within each SOA project, a common practice is to deploy portions of the SOA software whenever they can pass integration testing and quality checking, or even to use a phased-release structure for production use. This phased release to production may occur instead of holding the project's SOA software in order to end the SOA project by deploying all of the project's SOA software at one time. Phased release also tends to stimulate more satisfactory and useful user feedback to the project while the project is still ongoing (Erl 2008).

Assistance on doing SOA implementations is available from several sources, in addition to assistance available from employees within the organization moving to SOA use. Some software vendors offer not only their SOA software for licensing, but also offer to contract to provide implementation services using their own or subcontracted personnel. Some software and system consulting firms offer to contract either to do SOA implementation projects, or to custom prepare SOA software and help install it. Customized advice on SOA implementations is also available on a contract basis, and generic advice is available at conferences and through some professional membership organizations. Some journals, magazines, and books provide coverage about aspects of doing SOA implementations. (See the above noted references for some examples of these.)

Because of the usual distributed character of SOA-implemented systems and their use of the Internet for communication, an increasing proportion of SOA implementations include more reliance on the usage of cryptography than has been the case in non-SOA systems. SOA-implemented systems that utilize cloud or SaaS resources tend to be among the heavier users of cryptography, and reflect that usage in how they perform their information systems governance and internal auditing.

3.4 Evaluation

The characteristics of SOA-implemented systems may or may not conform to the characteristics of software maintenance. The bullet points in the two preceding sections of this paper lend themselves to a tabular qualitative summarization of the amount of conformity, where the rows (identified in the leftmost column) list the maintenance characteristics, and the columns to the right list the SOA-implementation characteristics and evaluation.

Assessing the conformity of two sets of characteristics to each other typically involves three main considerations. The first consideration is ontological in character. People from different backgrounds use their own familiar vocabularies in describing and defining entities, concepts, processes, practices, and situations. Having different names for things does not change the things and does not change same things into non-same things, or the non-same things into the same things. The second consideration is granular in character. Sometimes different descriptions or definitions reflect differences in the level of granularity or abstraction. This situation is like the classic blind men and the elephant. The third consideration is conceptual in character, and often involves time, scope, and realism. For example, an automobile may sometimes be moving rapidly, may sometimes not be moving at all, and may sometimes be moving at a rate in between these two states.

The results of assessing the conformity of the two sets of characteristics is shown in the four tables included in this paper. There, the “N” or “no” entries indicate where there is no or only minority conformity, usually because maintenance characteristics do not dominate. The “Y” or “yes’ entries indicate where there is full or major conformity, usually because the maintenance characteristics do dominate. The “Q” entries indicate that the conformity appears to be moot or open to reasonable question depending upon the local interpretation of the 14764 and other relevant standards, and/or upon the local SOA practices.

Because under some circumstances there may be some differences in the presence or absence of conformity, separate consideration is given to four situations.

Consider first an organization’s pilot (initial) SOA project, which is usually done in either of two major ways. One way is to incorporate the use of wrappers and/or middleware (such as an ESB) as part of the SOA project. In that situation, the resulting conformity among the SOA and software maintenance characteristics occurs as shown in Table 2.

Table 2 Pilot SOA Project with Wrappers and/or Middleware, such as ESB

Maintenance Characteristics	Software-Oriented Architecture (SOA) Characteristics					
	Project	Bounds	Replace	Wrapper	Distributed	Resource
Existing Software	Y	Y	Y	Y	Y	Y
Components	Y	Y	Y	Y	Y	Y
Software Maintenance Life Cycle	Y	Y	Y	Y	Y	Y
Artifacts	Y	Y	Y	Y	Y	Y
All types	Y	Y	Y	Y	Y	Y

Table 2 reflects the usual practice of focusing the pilot SOA project upon a small portion of the full extent of the existing functionality or services potential of an organization’s entire (enterprise) information system. The usual practice is not to tackle some new functionality, but to confine the

scope instead to some specific well-understood functionality (usually business processes) that the organization currently uses, but would like to have improved. Hence, that implementation's replacement by a SOA version modifies the existing information system, either by enhancing it, or adapting it to satisfy the changed conditions arising from the use of SOA (Chapin, Hale, et al. 2001).

The use of wrappers and/or middleware (such as the incorporation of an ESB) as part of the pilot project enables tying some of the parts of the existing information system into the SOA system instead of replacing and retiring them completely. This potentially reduces the cost needed of modifying existing software to incorporate web services. The usual practice in a pilot SOA project is for the organization and the personnel involved in the project to make a record of what was learned from the SOA project experience, to assist in future personnel doing any subsequent SOA projects for the organization. The personnel may be not only the participating employees of the organization, but also any vendor or contractor personnel and others who may have participated in the pilot SOA project.

For contrast, now consider doing the pilot SOA project in the original manner with web services and XML, but with scant or no use of middleware (such as an ESB), and with scant or no use of wrappers for the affected portions of an organization's existing information system. Table 3 shows the resulting conformity among the characteristics.

Table 3 Pilot SOA Project Without Wrappers and/or Middleware

Maintenance Characteristics	Software-Oriented Architecture (SOA) Characteristics					
	Project	Bounds	Replace	Wrapper	Distributed	Resource
Existing Software	Q	Y	N	N	N	N
Components	Y	Y	Y	N	Y	Y
Software Maintenance Life Cycle	Y	Y	Y	Y	Y	Y
Artifacts	Y	Y	Y	Y	Y	Y
All Types	Y	Y	Y	Y	Y	Y

Like Table 2, Table 3 reflects the usual practice of focusing the pilot SOA project upon a small portion of the full extent of the functionality or service potential of an organization's entire information system. Table 3 also is consistent with the usual practice of not tackling some new functionality, but instead sticking with some specific well-understood functionality (usually business processes) that the organization currently uses but would like to have improved.

In Table 3, the Q in the *Existing Software* row notes the role of local choice. Some organizations opt to create new versions and retire existing software. Others in the same situation opt to modify existing software to transform it to the classic web services SOA approach. The five N cells in Table 3 reflect that the classic approach replaces some existing software with web services versions and retires some existing software, thus incurring some cost.

As with the Table 2 situation, the usual practice in a classic pilot SOA project is to make a record of what was learned from the experience in order to assist the personnel doing any subsequent SOA projects. The personnel involved may be not only the participating employees of the organization, but also any vendor or contractor personnel who may have participated in the pilot SOA project.

A pilot SOA project is typically followed by other SOA projects to increase the scope of the information systems utilizing the SOA approach. This path is the one usually taken because it is commonly regarded as the path to reap in fuller measure the lower cost benefit of using SOA in the organization. The organization again has the choice of implementing SOA using wrappers and/or middleware (such as an ESB). Choosing this option leads to the conformity of characteristics summarized in Table 4.

Table 4 Subsequent SOA Projects with Wrappers and/or Middleware, such as ESB

Maintenance Characteristics	Software-Oriented Architecture (SOA) Characteristics					
	Project	Bounds	Replace	Wrapper	Distributed	Resource
Existing Software	Y	Y	Y	Y	Y	Y
Components	Y	Y	Y	Y	Y	Y
Software Maintenance Life Cycle	Y	Y	Y	Y	Y	Y
Artifacts	Y	Y	Y	Y	Y	Y
All Types	Y	Y	Y	Y	Y	Y

The conformity reported in Table 4 arises primarily from two sources. One is the extension of the use of wrappers and/or middleware (such as ESBs) to accomplish changes in the organization's information systems. The second is that each subsequent project builds upon and alters an existing software system that is the product of the prior SOA projects in an organization. This is of critical importance in the striving to obtain the full benefits claimed for the preferential reliance upon SOA software in the full panoply of an organization's information systems. This entails an adaptive modification to those systems (Chapin, Hale, et al. 2001, International Organization for Standardization 2006). An organization's objective for making such adaptive modifications usually is to try to realize the claimed SOA benefit of making its information systems more easily, quickly, and cost-effectively adaptable to the changing conditions in which an organization operates. Such agility in meeting an organization's needs must not necessarily happen as SOA projects—some organizations also extend SOA use as preventive actions to soften the hit of likely future cost increases.

Also, organizations may take advantage of extending SOA use in order to go beyond the pilot SOA project limitation of sticking only to currently implemented functionality in the information systems. Such organizations tend to regard subsequent SOA projects as an opportunity to also perform enhancive maintenance to add new or improved functionality to an organization's information systems (Chapin, Hale, et al. 2001). As was noted previously for pilot SOA projects, the usual practice in subsequent SOA projects is to learn from the experience. The personnel involved in the subsequent projects not only include the participating employees of the organization, but also any include vendor or contractor personnel who may have participated in SOA an organization's SOA projects.

As previously noted, a pilot SOA project is typically followed by other SOA projects as a path to more fully realize the benefits of using SOA within the organization. Even if not done in the pilot project, in the subsequent projects the organization may opt for taking the classic SOA approach of Web services and XML with scant use of wrappers and/or middleware. Choosing this option leads to the conformity of characteristics summarized in Table 5.

The conformity reported in Table 5 arises primarily from two sources. One is the alteration or replacement of existing software to adapt the retained existing software to fit with the existing SOA implementation, as software retirements occur more frequently than in the situation described in Table 4. The other is that each subsequent project is building upon and/or changing one of more existing software systems that are the products of the prior SOA projects in an organization. Software interfaces typically receive additional alterations to strengthen interoperability. This is of critical importance to an organization striving to more fully realize the benefits claimed from implementing SOA. Such an organization extends its SOA system implementations by making both enhancive and adaptive modifications to those implementations in an attempt to achieve more agility in adapting to changing requirements encountered by an organization (Chapin, Hale, et al. 2001). Some organizations also use projects extending SOA as preventive actions to try to help limit future cost increases.

Table 5 Subsequent SOA Projects Without Wrappers and/or Middleware

Maintenance Characteristics	Software-Oriented Architecture (SOA) Characteristics					
	Project	**Bounds**	**Replace**	**Wrapper**	**Distributed**	**Resource**
Existing Software	Y	Y	Y	Y	Y	Y
Components	Y	Y	Y	Y	Y	Y
Software Maintenance Life Cycle	Y	Y	Y	Y	Y	Y
Artifacts	Y	Y	Y	Y	Y	Y
All Types	Y	Y	Y	Y	Y	Y

Also, organizations often use the subsequent SOA projects to go beyond the pilot SOA project limitation of sticking only to currently implemented functionality in the information systems. Because organizations using the original web-services approach to SOA tend to have more geographically distributed information systems, they tend to regard subsequent classic SOA projects as an opportunity for doing enhancive maintenance to add new or improved functionality to an organization's distributed information systems (Chapin, Hale, et al. 2001). As was noted previously for pilot SOA projects, the usual practice in subsequent SOA projects is to learn from the experience. The personnel involved in the subsequent projects may not only include participating employees of the organization, but also vendor or contractor personnel who may have participated in an organization's previous SOA projects.

3.5 Discussion

The "service" in service-oriented architecture (SOA) originally just meant web service (Chapin 2007), but that narrow usage has been widely ignored in favor of broader usages. In some literature published in the computer field, such as the information technology infrastructure library (ITIL), a service is what one portion of an organization, aided by portions of the organization's implemented software, does to facilitate or support what one or more other portions of that organization do (Case and Spaulding 2007, Jones 2005). An example is providing data about specific customers or clients. Still broader is the normal language usage as defined in English language dictionaries; for example, "The physician gave me excellent service." In this paper, such normal language usage is not denoted.

As has been noted in the published literature, "service" is not a name identifying a unit of computer software, or a portion of a software-implemented system (Chapin 2008). A unit of computer

software (such as a routine or a procedure) when executed may perform a function (for example, encrypting a vendor record) that may be regarded as part of providing a service satisfying some person's identifiable need in a particular computer-using organization. And has been noted in the published literature, software evolution (such as going from using non-SOA software to using SOA software) is usually accomplished by doing software maintenance (Chapin, Hale, et al. 2001).

As noted earlier in this paper, an organization rarely focuses its pilot SOA project on business processes or functions that are new to the organization, instead focusing on modifying existing in-production implementations of existing business processes or functions so that they may adapt their in-production implementations to qualify as SOA implementations. The initial (and usually more expensive) way to do this is to replace existing in-production implementations with new implementations, which include classic SOA-adapted functionality in the pilot and subsequent SOA projects. (See Table 3 and Table 5 in Section 3.4.)

A usually less-expensive initial way to do this in the pilot SOA project is to provide wrappers for the existing targeted functionality, and provide interoperability by adding middleware, with a common example being an ESB. This adds new software to the existing in-production implementation, while leaving the business processes or functions intact, but adapts the characteristics of the system implementation to qualify as a SOA. Such modifications satisfy the definition of evolution or maintenance (Chapin, Hale, et al. 2001, International Organization for Standardization 2006). Also, such modifications usually result in less system improvement for system users, because the adaptive modifications done are less numerous and less extensive. In contrast, major and numerous system improvements usually are implemented when enhancive modifications are the focus of the SOA projects. Subsequent SOA projects typically continue to use the approach used in the pilot SOA project. (See Table 2 and Table 4 in Section 3.4.)

In either alternative, the modified and/or added software also increases the pool of software that eventually will need software maintenance work in the future, such as for enhancive maintenance (Chapin, Hale, et al. 2001). This future potential, combined with the software effects of introducing and extending SOA usage, appears very likely to increase the amount of software maintenance needed in the future. This is because of the software maintenance done in the pilot and subsequent SOA projects that have resulted in a net increase in the total size and complexity of the enterprise information systems.

This increased potential burden of future maintenance also directs attention to the common claim in the literature about the potential contribution of SOA to improved agility in making future changes to information systems to meet the changing needs of the using organization (Freivald 2008). Making changes to information systems may be system evolution, and such evolution is usually achieved by doing software maintenance.

Performing SOA software projects may help realize the claim of improved agility, in spite of SOA projects being primarily software maintenance; but realizing this claim might come from the same sources as do such claims of improved agility from using agile software processes. As one of the signers of the Agile Manifesto has noted in literature on agile methods, agile software processes are mostly software maintenance processes (Beck 2000)—a reality that is not commonly trumpeted by advocates of agile processes. Furthermore, as noted in Section 3.1, a claimed benefit from using SOA is a reduction in the amount (and by implication, of the proportion) and

the cost of software maintenance done. That position appears inconsistent with the findings reported here that the characteristics of doing SOA implementations are primarily the characteristics of doing software maintenance.

3.6 Conclusion

In summary, the evaluation reported here found that SOA implementations are mostly software maintenance implementations. Nearly all SOA projects are done primarily by doing software maintenance processes such as expounded in the ISO/IEC 14764 standard or the IEEE Std 14764-2006 standard, except sometimes for parts of a pilot SOA project. This is a potentially useful observation that points to additional research opportunities:

1. Does it partially explain the difficulties some organizations have experienced in achieving budget, time schedule, and/or requirements satisfaction in their SOA projects? The organizations may have been trying to manage them as if they were software development projects. As the 14764 standard notes and further emphasizes by its very existence, maintenance is not the same as development.
2. To what extent is the SOA claim that agility is improved in meeting organizations' changing information systems needs, when the organization is a SOA user?
3. To what extent is the observed agility change in meeting the changing needs of an organization, related to whether the information systems change is done as a SOA project, or not? To what extent is the use of web services, middleware, and wrappers related to the amount of agility change?
4. For the enterprise that has done at least one SOA project, how has the size of that enterprise's total information systems changed? How does this change vary with the number of SOA projects done, and with the changes in the total size of other like enterprises' information systems?
5. How do the frequency, cost, and extensiveness of maintenance differ for SOA and non-SOA implementations doing comparable functions in different but similar organizations?
6. For organizations that have SOA-implemented software in use, how does the frequency and cost of doing the various types of maintenance (such as enhancive or corrective) compare for SOA and pre-SOA time periods? This is a difficult comparison, since environmental conditions will have changed.
7. How has the SOA-using organization's total proportion of maintenance personnel hours changed from pre-SOA to with SOA? Because organizations and their environments change with the passage of time, this is a difficult comparison to make.
8. In a case study of some specific SOA-using organization, what types of maintenance were actually done and in what proportions in its SOA projects?
9. In a case study of some specific SOA-using organizations, how does the personnel-hour usage vary for specific types of maintenance in the subsequent SOA projects, and in the maintenance of the SOA software done in response to user requests for modifications of the SOA-implemented software used by the requester?
10. Using the ISO/IEEE standard as the context, how much has SOA usage reduced the burden of software maintenance for the SOA-using organizations, by varied times since they first began their pilot SOA projects?

References

Baer, Tony. "SOA: Building the Roadmap." *ZapThink.* June 2007. http://www.zapthink.com/2007/06/29/soa-building-the-roadmap/ (accessed January 2010).

Beck, K. *Extreme Programming Explained: Embrace Change.* Boston: Addison-Wesley, 2000.

Case, G., and G. Spaulding. *Continual Service Improvement ITIL Version 3.* London: The Stationery Office, 2007.

Chapin, Ned. "Integrating BPM with SOA." *Proceedings of the 2nd Workshop on SOA-Based Systems Maintenance and Evolution* . Los Alamitos: IEEE Computer Society, 2008. 3.

—. "Service Granularity Effects in SOA." *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering.* Redwood City: Knowledge Systems Institute Graduate School, 2008. 506-511.

—. "Value Focused Research Directions on Service-Oriented Architecture Applications." *Proceedings of the 1st International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems.* IEEE Computer Society, 2007. 2.

Chapin, Ned, J. E. Hale, K. Md. Khan, J. F. Ramil, and W. G. Tan. "Types of Software Evolution and Software Maintenance." *Journal of Software Maintenance and Evolution*, 2001: 3-30.

Erl, T. *SOA: Principles of Service Design.* Boston: Pearson Education, 2008.

Ferris, C., and J. Farrell. "What Are Web Services?" *Communications of the ACM* 46, no. 6 (2001): 31.

Freivald, J. "Why Business Managers Should Care About Service-Oriented Architecture." *iWay Software.* June 2008. http://www.ithound.com/v3/download/2073 (accessed January 2010).

Geminiue, K. "A Service-Oriented Approach to Business Rules Development." In *SOA Best Practices: The BPEL Cookbook*, by Harish Gaur, 15. Redwood City: Oracle Corporation, 2008.

International Organization for Standardization. "Software Engineering - Software Life Cycle Processes - Maintenance, ISO/IEC 14674." *International Organization for Standardization.* 2006. http://www.iso.org/iso/home.htm (accessed January 2010).

Jones, S. "Toward an Acceptable Definition of Service." *IEEE Software* 22, no. 3 (2005): 87-93.

Josuttis, N. *SOA in Practice.* Sebastopol: O'Reilly Media, 2007.

Knorr, E., and O. Rist. "10 Steps to SOA." *InfoWorld* 45 (2005): 23-25, 28, 30, 32-34, 36, 38, 40, 42, 45-46, 48-52.

Lewis, Grace, Edwin Morris, D. R. Smith, Soumya Simanta, and Lutz Wrage. "Common Misconceptions about Service-Oriented Architecture." *CrossTalk* 20, no. 11 (2007): 27-30.

Ma, K. J. "Web Services: What's Real and What's Not." *IT Professional* 7, no. 2 (2005): 14-21.

Mahmood, Zaigham. "Service Oriented Architecture: Potential Benefits and Challenges." *Proceedings of the 11th World Scientific and Engineering Academy and Society (WSEAS) International Conference on Computers.* Stevens Point: World Scientific and Engineering Academy and Society, 2007. 497-501.

Manes, Anne T. "Enterprise Service Bus: A Definition." *Application Platform Strategies: In-Depth Research Overview.* Midvale, UT, October 2007.

Nitu. "Configurability in SaaS (Software as a Service) Applications." *Proceedings of the 2nd Annual Conference on India Software Engineering Conference.* Pune: Association for Computing Machinery, 2009. 19-26.

Oracle. "Strategies for SOA Success." *Align Journal.* 2005. http://www.inst-informatica.pt/servicos/informacao-e-documentacao/biblioteca-digital/arquitectura-e-desenvolvimento-de-aplicacoes/soa/strategies-for-soa-success.pdf (accessed January 27, 2010).

Reddy, V. K., A. Dubey, S. Lakshmanan, S, Sukumaran, and S. Sisodia. "Evaluating Legacy Assets in the Context of Migration to SOA." *Software Quality Control* 17, no. 1 (2009): 51-63.

Schepers, T. G. J., M. E. Iacob, and P. A. T. van Eck. "A Lifecycle Approach to SOA Governance." *Proceedings of the 2008 ACM Symposium on Applied Computing.* New York: Association for Computing Machinery, 2008. 1055-1061.

Sedayao, J. "Implementing and Operating an Internet Scale Distributed Application Using Service Oriented Architecture Principles and Cloud Computing Infrastructure." *Proceedings of the 10th International Conference on Information Integration and Web-Based Applications and Services.* New York: Association for Computing Machinery, 2008. 417-421.

Willis, T. *BPEL 100 Success Secrets, Business Process Execution for Web Services, the XML-based Language for the Formal Specification of Business Processes.* London: Emero Pty Ltd, 2008.

4 ASMEM: A Method for Automating Model Evolution of Service-Oriented Systems

S. Khoshnevis, P. Jamshidi, A. Nikravesh, A. Khoshkbarforoushha, R. Teimourzadegan, F. Shams (Shahid Beheshti University GC, Iran)

Abstract

A service model is the key work product of the service-oriented solution life cycle, in which localizing the effects of business changes could reduce the cost of evolution and maintenance. Practitioners in service-oriented computing strongly desire automated methods, tools, and environments that accommodate changing business requirements both cost-effectively and in an agile way. To achieve this goal, this paper proposes an automated approach for service model evolution named Automated Service Model Evolution Method (ASMEM), which enjoys two salient characteristics: a) it localizes the business model changes to the service model in a traceable manner; and b) it propagates incremental changes instead of re-transforming the entire changed business model. To achieve these characteristics, two steps are proposed: first, several types of change scenarios to the input business model are explored; and second, evolution management steps are provided for each scenario. In fact, these steps represent how these changes should be propagated to the service model, based on quantitative criteria. The paper also discusses the preliminary implementation of Automated Service-Oriented Modeling Tool (ASOM-Tool), which is developed to supply automation and traceability features for modeling and maintaining service-oriented solutions.

4.1 Introduction

Evolution is recognized as an extremely labor-intensive activity in the software life cycle (Boehm and Papaccio 1988), particularly in today's competitive businesses. Because any system is exposed to many different change conditions, most of which emerge from the business itself (Bennett and Rajlich 2000, Lehman and Ramil 2003), they have to be flexible enough to encounter these changes, and dynamically adapting (Shaw and Clements 2006) to them would help to maximize the benefit to each user. Since service-oriented solutions have a cohesive relation with business (Arsanjani, et al. 2008, Wahli, et al. 2007), they have to be capable of accommodating unanticipated business changes cost-effectively and in an agile way. Therefore, the theme of maintenance and evolution of service-oriented systems is in the focus of today's research (MESOA 2009).

Service-oriented modeling is the discipline of modeling business and systems for the purpose of designing and specifying service-oriented solutions within a service-oriented architecture (Jamshidi, Sharifi and Mansour, To Establish Enterprise Service Model from Enterprise Business Model 2008). According to IBM SOMA (Arsanjani, et al. 2008), the service-oriented modeling life cycle comprises three main phases: service identification, service specification, and service realization. In order to effectively and efficiently produce a service model, service modeling activities should be supplied with automation features in a systematic and model-driven fashion (Wahli, et al. 2007). In this regard, the authors proposed Automated Service Identification Method (ASIM) (Jamshidi, Khoshnevis, et al. 2009) and Automated Service Specification Method

(ASSM) (Jamshidi, Koshnevis, et al. 2009, Jamshidi, Sharifi and Mansour 2008) to automatically populate the details of the service model.

The service model work product is the core artifact of service-based solutions and is used as an essential input to implementation and test activities. Therefore, localizing and applying the effects of business changes in the service model work product by leveraging design and dependency information (Hearnden 2007) could reduce the cost of maintenance. Due to this issue, there is truly a high motivation for developing an automated method for service model evolution. Therefore, the main objective of this research work is to investigate incremental (vs. re-transformation) change propagation through service modeling activities in the service-oriented solution life cycle, as an enabling mechanism towards automated service-oriented system maintenance and evolution.

In this regard, we propose an automated method for service model evolution, which is based on the notions and concepts of an automated service-oriented modeling life cycle through ASIM and ASSM. In other words, realizing a fully-automated service-oriented modeling life cycle requires another building block to support model evolution features, namely Automated Service Model Evolution Method (ASMEM). ASMEM enjoys two salient characteristics: a) it localizes the business model changes to the service model in a traceable manner; and b) it propagates incremental changes instead of re-transforming the entire changed business model.

Because any possible approach for fully-automated service-oriented modeling must be supported by a tool, we introduce the Automated Service-Oriented Modeling Tool (ASOM-Tool) (Nikravesh, et al. 2009) which provides a model-driven environment for capturing design and dependency information through the related models and corresponding transformations. Moreover, ASOM-Tool supports model evolution features, which can efficiently redesign the service model in case of business model changes. This means that by adopting the tool, consequential changes induced by the maintenance activities in the systems can be accommodated automatically, incrementally, and efficiently, reducing the cost of software maintenance and evolution.

Because limited research has occurred on the model-driven software evolution in the context of service-oriented solutions, we briefly introduce most relevant work in Section 4.2. Regarding our previous research work, several concepts which have been reused in this work are presented in Section 4.3. The position of model evolution in the service-oriented modeling framework, several change scenarios in service-oriented solution maintenance, our proposed solution to accommodate them, and its implementation in a tool are described in Section 4.4. The evaluation of the contribution has been conducted through a case study described in Section 4.5. Finally, the most well-known challenges of software evolution that we have tackled in this work are discussed in Section 4.6, and concluding remarks are presented in Section 4.7.

4.2 Related Work

While substantial research from both industry and academia has been devoted to software evolution and maintenance, there has been limited exploration of these concepts in service-oriented computing. In addition, although some techniques such as service identification, concept location, and service testing were introduced to support maintenance and evolution of service-oriented systems (MESOA 2009), work on automated model-driven software evolution in service-oriented systems is nonexistent in the literature, to the best of our knowledge. However, this section briefly reviews the closest work accomplished over decades in (automated) software evolution.

There are several techniques to accomplish (semi-) automated software evolution, ranging from heavyweight, formal methods (Yang and Ward 2003, Wiels and Easterbrook 1998), to lightweight, ad hoc approaches (Rajlich 1997). Some of the more well-established techniques include reengineering (Yang and Ward 2003), change impact analysis (Arnold 1996), techniques based on category theory (Barr and Wells 1990), and techniques based on meta-programming (Mens and Tourwe 2001, Tourwe and Mens 2003). The underlying foundation of these techniques is the proper exploitation of metadata. Reengineering is dominated by the recovery of design information, whereas change impact analysis depends heavily on dependency information. Category theoretic techniques rely on implicit categorical metadata, while meta-programming directly exploits metadata.

4.3 Basic Concepts

In this section, definitions of relevant terminology and its implications are presented. For more details, see (Jamshidi, Sharifi and Mansour 2008, Jamshidi, Khoshnevis, et al. 2009).

In order to utilize the business model in this work, the lowest level of its dimensions that are applicable in a conceptual view should be adopted. Referring to the literature of software engineering, an *Elementary Business Process (EBP)* is defined as *"a process performed by one person in one place at one time which adds significant value and which leaves data in a consistent state"* as the lowest level of enterprise business process. In addition, "*an enterprise domain model should be decomposed to the extent that each entity is created only in one business process and used in the other needed processes*", and in literature, it is called *business entity*.

Definition 1: A *Business Entity (BE)* can be defined as $BE = \{n, A, R\}$, where n is the name of the business object, A is the set of attributes, and R is the set of relationships between BE and other business entities.

Definition 2: An *Elementary EBP* can be defined as $EBP = \{n, (BE_j, sr)\}$, where n is the name of the elementary business process, BE_j is the j^{th} business entity which semantically relates to the corresponding EBP. $sr \in \{"C", "R", "U", "D"\}$ is the type of semantic relationship between EBP and BE_j.

In the present work, in order to have an appropriate business model as an input of the method, we use the EBPs as rows and BEs as columns of the matrix, which is called a CRUD matrix[2] (Jamshidi, Sharifi and Mansour 2008). Therefore, a cluster of the matrix could be representative of an abstraction level (Business Capability), in which EBPs act as their behavioral elements and its BEs act as structural elements. These clusters are business-aligned entities, and therefore are at a much higher level of abstraction than are objects and components.

Definition 3: A CRUD matrix can be defined as $M = \{(EBP_i, BE_j)\ i=1...\#row, j=1...\#column\}$, where EBP_i is the i^{th} EBP and BE_j is the j^{th} BE. *#row* is the number of *EBPs* and *#column* is the number of *BEs* in the model.

Many definitions have been proposed for services, but we have found the ones below to be proper. A *service* is a well-defined, encapsulated, reusable, business-aligned capability. A *service op-*

[2] CRUD: Create, Read, Update, Delete

eration is the elementary part of a service and specifies the associated inputs, purpose (function, duty or obligations), and outputs (artifacts, products, outcomes, or deliverables) (Arsanjani, et al. 2008).

Definition 4: A software service can be defined as *S= {n, I, Msg, RS}*, where *n* is the name of the service, *I* is the set of operations associated with *S*, *Msg* is the set of messages corresponding to *S*, and *RS* is the set of relationships between *S* and the other services in the service model. Referring to the CRUD matrix, $I \subset EBP$ is the set of corresponding EBPs, and $Msg \subset BE$ is the set of corresponding BEs (Wiels and Easterbrook 1998).

An enterprise service model is a model of the core elements of a SOA and is used as an essential input to activities in implementation and testing of software solutions. The service model is an abstraction of the software services implemented within an enterprise and supporting the development of one or more service-oriented solutions. It is a comprehensive and composite artifact encompassing all services, providers, specifications, partitions, messages, collaborations, and the relationships between them (Arsanjani, et al. 2008).

Definition 5: An *Enterprise Service Model* can be defined as *ESM = {S, Pr, Spec, Prt, Msg, R}*, where *S* is the set of services, *Pr* is the set of providers of the services, *Spec* is the set of specifications in a form of contract between what consumers need and what providers provide. In addition, *Prt* is set of partitions in order to manage the solution, *Msg* is the set of messages forming the underlying data model of the solution, and *R* is the set of relationship between services.

Definition 6: Suppose we have identified *n* services *{S1, S2, … , Sn}* from a business model and form service set *S*. We can use the average of each technical metrics (TM_j) as an objective function of the problem as follows:

$$\frac{1}{n}(Max(\sum_{i=1}^{n} TM_j(S_i))),\ \frac{1}{n}(Min(\sum_{i=1}^{n} TM_{j'}(S_i)))$$

The consolidated objective function of the problem can be the multiplication of the individual objective functions. Therefore, the service identification problem (SI) can be mathematically delineated as follows:

$$SI\colon EBM \rightarrow ESM$$

$$S=\{S_1, S_2, \dots, S_n\}$$

$$\text{Max } Z(S),$$

$$Z = \prod_j \sum_{i=1}^{n} TM_j(S_i) \Big/ \prod_{j'} \sum_{i=1}^{n} TM_{j'}(S_i)$$

We have adopted three technical metrics to derive the appropriate objective function represented as follows. The comprehensive description of how to derive the Z is presented in (Jamshidi, Khoshnevis, et al. 2009).

$$Z(S) = \frac{\sum_{i=1}^{n} G(S_i)^{3} * \sum_{i=1}^{n} \text{Cohesion}(S_i)^{2}}{\sum_{i=1}^{n} \text{Coupling}(S_i)^{2}}$$

where G(S), Cohesion(S), Coupling(S) stand for granularity, cohesion, and coupling of the service S respectively.

Definition 7: In software engineering literature, the terms *evolution* and *maintenance* are virtually synonymous. Occasionally, maintenance refers only to particular kinds of evolution, but, in general, the two terms are used interchangeably (Bennett and Rajlich 2000, Hearnden 2007). We shall use maintenance to refer to general post-delivery activities, and evolution to refer to consequential change. A change to a model in a model-driven system may require other changes to be made to other models.

Definition 8: The obvious approach for effecting consequential change induced by a transformation relationship $t = T(s)$ is simply to execute the transformation again, regenerating the output models, as shown in Figure 4.

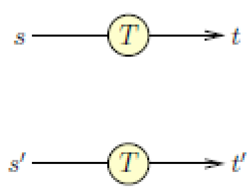

Figure 4 Re-Transformation (Hearnden 2007)

A fundamental limitation of *re-transformation* strategies is that they require a complete re-execution of the transformation. In contrast, *incremental* strategies address this problem by handling the consequential change directly. Where the focus of merged re-transformation is the difference of the transformation outputs, the focus of an incremental strategy is the transformation of the difference of the inputs, as shown in Figure 5.

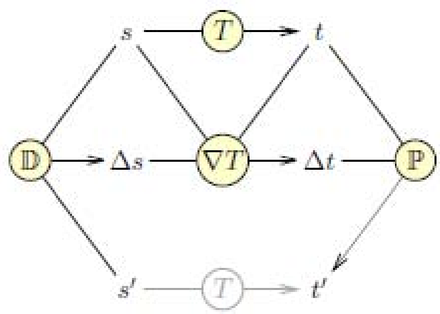

Figure 5 Incremental Change Propagation (Hearnden 2007)

4.4 Evolution Management in our Service-Oriented Modeling Framework

In this section, we will provide our solution for automating management of service model evolution in our service-oriented framework. For this purpose, we will briefly discuss evolution management issues in model-driven systems and the position of ASMEM in our framework. Thereaf-

ter, we will provide main evolution scenarios and steps on how to manage each scenario in ASMEM, and the tool support (ASOMT) will be explained.

4.4.1 Evolution in Model-Driven Systems

A change to a model in a model-driven system may propagate to other models, which is referred to as consequential change (Hearnden 2007). If the reason for a consequential change can be explained objectively (that is, without domain or semantic knowledge), then there is the potential to automate its derivation and application. The only objective reasons for consequential change in a model-driven system are the preservation of meta-model and transformation relationships. Any other kind of consequential change requires specialized domain knowledge, and thus cannot be automated without that knowledge.

In general, meta-model relationships cannot be usefully maintained automatically; however, their maintenance may be assisted by automation. But the maintenance of transformation relationships is completely automatable because they have machine-interpretable semantics.

As mentioned in Definition 8, there are non-incremental maintenance strategies such as re-transformation, as well as incremental strategies such as delta-transformation (Hearnden 2007). The latter was used as a basis for model evolution management in ASMEM.

4.4.2 Model Evolution in the ASOM Framework

Using traditional solution development environments to develop service-oriented solutions usually causes that architects who work directly with the business models to get involved in deriving the solution models by utilizing prescriptive guidelines such as SOMA (Arsanjani, et al. 2008) as depicted in Figure 6. In this regard, the authors proposed Automated Service-Oriented Modeling Framework (ASOMF) (Jamshidi, Khoshnevis, et al. 2009), which consists of a cohesive assembly of methods, techniques, tools, and content in order to support service-modeling activities, from business modeling to the generation of implementation artifacts. ASIM, ASSM, and ASOM-Tool are parts of this framework.

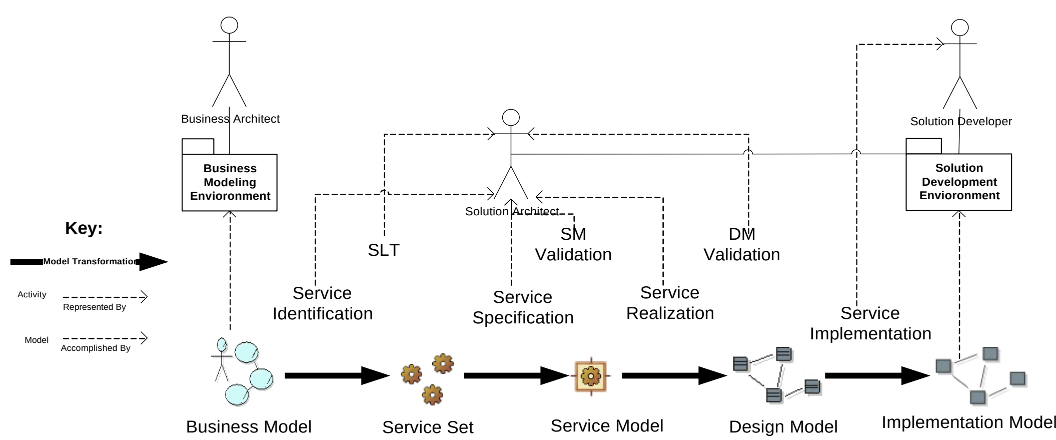

Figure 6 Service-Oriented Solution Life Cycle without Using the ASOM-Tool

ASIM is used to automatically identify the candidate services set as the main architectural elements of the service model work product, with respect to specified technical metrics. In this me-

thod, service abstractions with acceptable technical metrics can be derived automatically from high-level business requirements and business process models. We define the CRUD matrix as the desired abstraction level. A cluster (Jamshidi, Sharifi and Mansour 2008) of the matrix would represent a business-aligned service. In this method, each candidate service would be identified based on an objective function named Z. (For more details, see (Jamshidi, Khoshnevis, et al. 2009).)

The ASSM method is intended for specifying the service model by deriving service model elements from the candidate services set.

In order to fully automate the life cycle, there is a need to introduce another method that is responsible for model evolution management. For this purpose, we propose the Automatic Service Model Evolution Method (ASMEM) in this paper.

ASMEM provides steps for handling every evolution scenario. The evolution scenarios are based on six categories of changes that may occur in the input model. This method and its implementation will be fully discussed in Sections 4.4.3 and 4.4.4 respectively.

ASOM-Tool provides an integrated environment that supports service modeling activities by providing automated features throughout the solution lifecycle. Its architectural model is depicted in Figure 7.

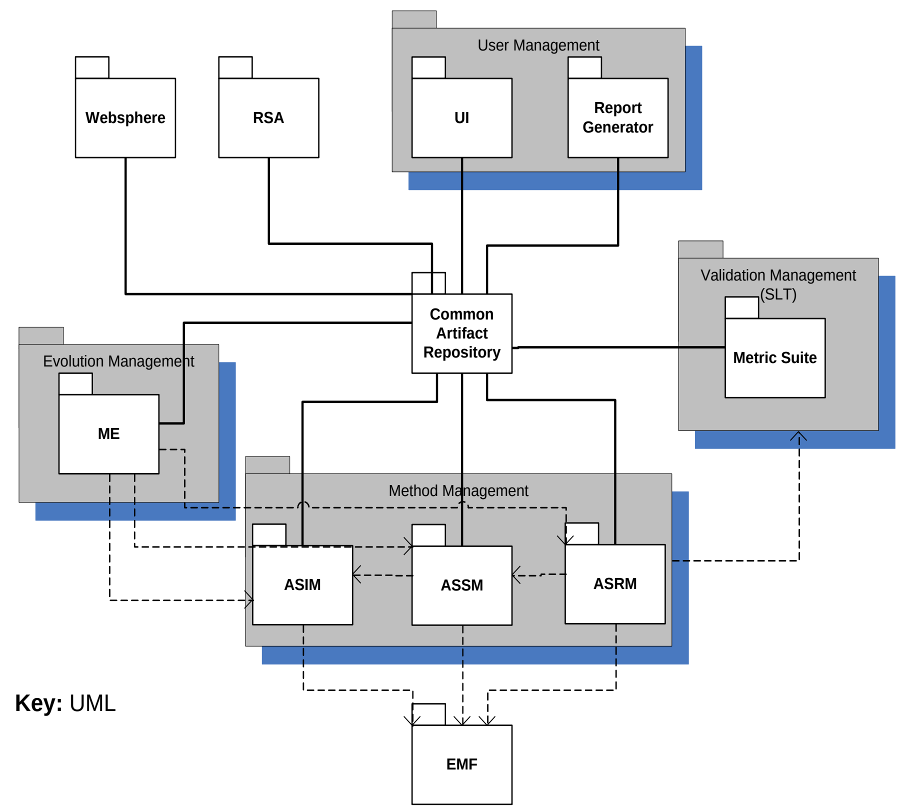

Figure 7 ASOM-Tool Architecture

4.4.3 Managing Evolution Scenarios in ASMEM

In order to enable automated software evolution, it is essential to understand what types of modifications are carried out in evolutionary tasks, rather than why those changes have been performed (Hearnden 2007).

Since ASIM and ASSM are based on the CRUD matrix, changes affecting its rows and columns lead to changes in services. Therefore, if EBPs or BEs of an enterprise are changed, the enterprise candidate service set should be identified again. The set of candidate services is crucial for generating and dealing with the service model in the ASSM stages (Jamshidi, Khoshnevis, et al. 2009). One solution to this problem is applying ASIM to the modified CRUD matrix; however, this approach takes much time and is costly. Another approach to solve the problem is categorizing possible changes in the CRUD matrix and proposing steps to reconfigure the candidate service set based on the type of the occurred change. Using the CRUD matrix, changes are located in the source model, and then using model transformations in the ASSM, the changes are propagated to the service model, as shown in Figure 8.

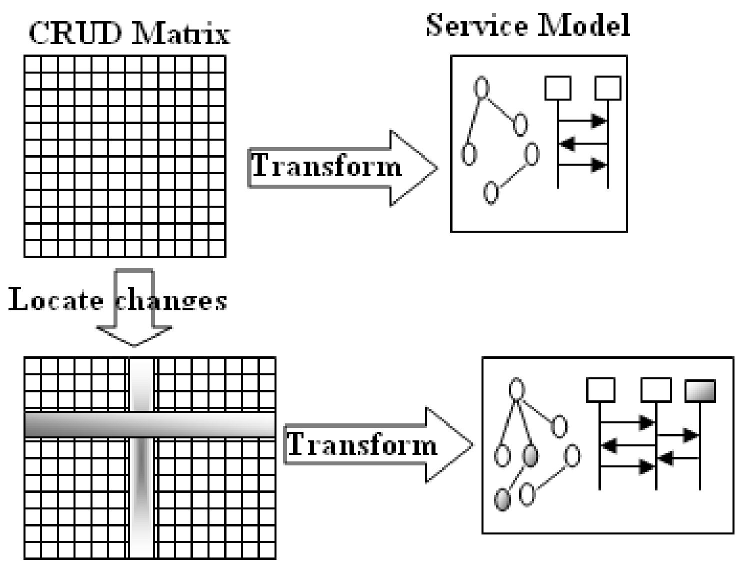

Figure 8 Changes are Located in the CRUD Matrix and Then Propagated Into the Service Model Through Transformation

There are six possible categories of changes in the CRUD matrix rows and columns. We have proposed steps to face these changes with proper reconfiguration of the candidate service set for each category as follows:

1. **Adding a new EBP to the CRUD matrix**. When a change to the business processes occurs in terms of adding a new EBP, we can handle it by considering it as a new operation for one of the identified enterprise services. For this purpose, we consider a new row added to the CRUD matrix and based on that, we check how many BEs are common between the new EBP and each of the services—which are clusters in the CRUD matrix. The service with the most common BEs will be the one to which the new EBP is added as a new operation, only if the objective function of the service is greater than the specified threshold.

2. **Adding a new BE**. A change to the business processes may occur in terms of adding a new BE. In this case, a new column is added to the CRUD matrix and should be positioned in a proper place somewhere between the other columns, where it maximizes the objective function (Z). Then based on the objective function one of the following actions may be taken:
   - The new BE is added to the service which is placed as a cluster at its northwest.
   - The new BE is added to the service which is placed as a cluster at its southeast.
   - A new service is defined to cover the new BE.
3. **Adding a new EBP and a new BE simultaneously**. This change can be applied to the CRUD matrix in two steps: the first step is adding a new BE to the CRUD matrix, and the second is adding a new EBP to it. Adding a new EBP can be influenced by the result of adding a new BE—that is why we first add the BE and then the EBP.
4. **Deleting a BE**. To delete a BE from the CRUD matrix, the first step is to verify the service which contains the BE. After deleting the BE, if there are still "C" actions in the service that uses the BE, we can simply delete the BE from the CRUD matrix. But if deleting the BE from matrix causes deleting all "C" actions of the service, the service no longer exists and based on Z, one of the three following actions would be taken:
   - Cells that remain after deleting the service would be added to the service placed in the northwest area of the matrix.
   - Cells that remain after deleting the service would be added to the service placed in the southeast area of the matrix.
   - Cells that remain after deleting the service would be considered as service channels between remaining services only.
5. **Deleting an EBP**. If an EBP includes a "C" action on a BE, then deleting the EBP causes deletion of that BE. Therefore, to delete an EBP from the CRUD matrix, the first step is to delete the BEs that would be deleted as a result of deleting the EBP. Then, the steps for the fourth category mentioned above can be applied to reconfigure the matrix. Finally, the EBP can be deleting resulting in the deletion of an operation of a service.
6. **Deleting a BE and an EBP simultaneously**. This change would be applied to the CRUD matrix as first deleting the EBP and then deleting the BE from the matrix.

4.4.4 Evolution Management in the Tool Architecture

We developed the ASOM-Tool (Nikravesh, et al. 2009) to implement the methods and guide service-oriented practitioners to design service-oriented solutions based on the mentioned service modeling methods in a traceable, reusable, configurable, and systematic way. The ASOM-Tool is a product, which is developed on top of the Eclipse Modeling Framework (EMF) to provide automation features for modeling service-oriented solutions. This product helps system architects to automatically accomplish the service-oriented modeling activities.

We used the ADD method (Bass, Clements and Kazman 2003) for creating the architecture of the ASOM-Tool. The architectural blueprint of the ASOM-Tool is shown in Figure 7.

There are four key modules located around a *common artifact repository* for storing intermediate and final results. This tactic helps us to keep track of the outputs of each module and facilitates

the integration of different modules, which increases the modifiability of the architecture. Responsibilities of the key modules are as follows:

- *User Management* manages user interaction with the system. There are two sub-modules, which are “user interface” and “report generator.”
- *Validation Management* is responsible for quantitative analysis of produced models, such as the service model and its elements.
- *Evolution Management* enables links between the solution artifacts (such as services and service components) and changing business models.
- *Method Management* implements the algorithms for ASIM, ASSM, and ASRM [10] according to the solution life cycle depicted in Figure 9.

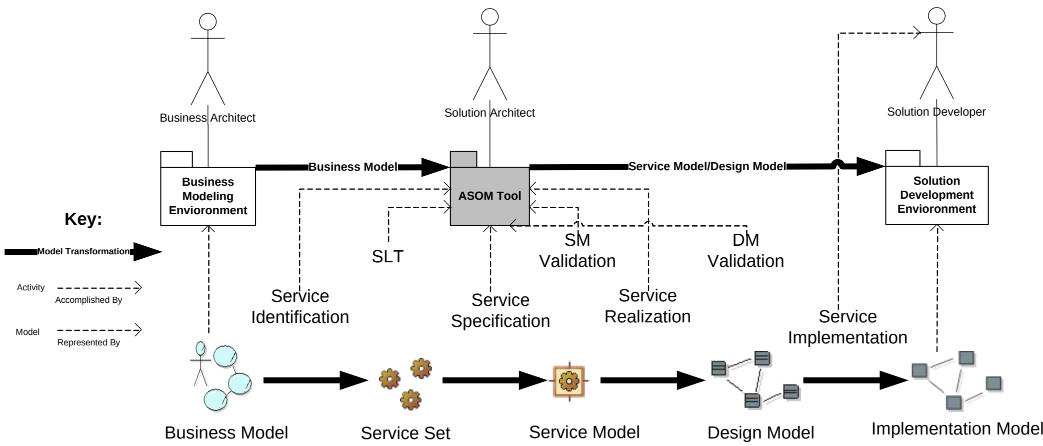

Figure 9 Using ASOM-Tool in the Service-Oriented Solution Life Cycle

The evolution management module deals with the common artifact repository and the method management module. The dependency between the evolution management module and the common artifacts repository refers to the need to maintain intermediate and final model evolution artifacts in a common environment that enables the tool to share them with other modules. The dependency between the evolution management module and the method management module is defined in terms of dependencies between the Model Evolution (ME) and the sub-modules inside the Method Management module. ASIM and ASSM use the CRUD matrix as one of their inputs; therefore any change in the CRUD matrix directly affects them.

The ASOM-Tool’s strategy for the incremental change process is an extension of a transformation engine (Method Management Module) called live transformation. Once the initial transformation has been completed, the engine listens for changes to the input models (business model), and maps them directly to changes in the output models (service model). Live transformation propagates the updates incrementally and only re-computes the affected parts of the transformation.

Using the model evolution capability adds several new functionalities to the basic ASOM-Tool:

- Adding new services to the service model.
- Removing an available service from the service model.
- Adding to or removing operations from services.

- Adding to or removing parameters (messages) from operations.

It is clear that these new functionalities increase flexibility and maintainability in reference to the service model base elements, which are mostly services and their operations and messages. Other service model elements are mostly derived once the services and their operations and messages are determined.

4.5 A Case Study

Verification of the proposed steps was done by studying examples and performing case studies, one of which is described below.

The selected enterprise for the study is the sales department of a sales and goods distribution company. In order to simplify the problem description and its scope, we limited the business scenario to main and general (and yet real) issues, and ignored very detailed exceptions.

The department receives orders in a daily manner from mostly regular and some casual customers. The orders are placed by a role: "salesperson." The salesperson delivers the orders to the sales-in-charge, who then passes the orders on to the sales clerk for processing. Processing orders includes checking customer's credit information, checking discounts for the goods ordered (if any), checking inventory, and calculating the final order price. When the customer approves the final price, the order is ready to be shipped. The storekeeper issues a receipt, having delivered the goods to the customer. The financial department representative issues a financial “notes receivable” when a payment is made.

Our business model is represented through a CRUD matrix and is clustered by means of ASIM (Jamshidi, Khoshnevis, et al. 2009), as shown in Table 6. We are going to apply changes in terms of adding and removing EBPs and BEs to and from the matrix.

Table 6 Original CRUD Matrix

	Customer	Credit	Account s Receivable Note	Discounts	Inventory	Warehouse Voucher
Add a customer	C	C				
Add an accounts receivable note	R	U	C			
Receive order	R	R				
Receive order	R					
Calculate discounts				R		
Check inventory					R	
Add discounts				C		
Add an item					C	
Add a warehouse voucher		R			U	C

In order to explain how the steps are applied, two examples for the first and the second change categories are provided. Suppose that we add a new EBP to this matrix as "Calculate price" (a change that fits in the first category mentioned in Section 4.4.3). The new matrix is shown in Table 7 based on the first category of steps in Section 4.4.3. The first step to handle this change is to place the new EBP in the right row of the CRUD cluster. As seen in Table 7, the new EBP has only one BE in common with Service 2. Therefore Service 2 is the first candidate for adding the new EBP. Also, the value of Z (represented in the second row of Table 8) for the new service is greater than the threshold (in order to avoid complicated computations, we have omitted the quantitative value of the parameters); therefore, this EBP will be added to Service 2.

Table 7 Adding an EBP to the CRUD Matrix

	Customer	Credit	Accounts Receivable Note	Discounts	Inventory	Warehouse Voucher
Add a customer	C	C				
Add an accounts receivable note	R	U	C			
Receive order	R	R				
Receive order	R					
Calculate discounts				R		
Add discounts				C		
Calculate price				R		
Add an item					C	
Add a warehouse voucher		R			U	C

Table 8 Values of Z for the Categories of Change Scenarios

Change Category	Z for Service 1	Z for Service 2	Z for Service 3	Total Z
-	89.55	19.56	23.0	1222.39
1	89.55	1.30	1.0	196.51
2a	121.5	2.48	19.6	547.3
2b	43.74	30.375	19.6	832.94
3	43.74	30.375	19.6	832.94
4	43.74	2.41	0.53	277.97
5	43.74	45.32	omitted	362.06
6	43.74	45.32	omitted	362.06

Now suppose we add a new BE, "Order," to the matrix in Table 7 (a change that fits in the second category). The new matrix will be the one shown in Table 9. The new BE should be placed in the right column of the CRUD matrix based on its "C" semantic relationship. As you can see, "creation" of the new BE is done by the "Receive order" elementary business process. Thus, this BE will be placed on the left side of Service 2.

As mentioned in Section 4.4.3, based on value of Z, the new BE would be added to Service 1 or Service 2. The third and fourth rows of Table 8 represent Z values, in the case of adding the new BE to Service 1 or Service 2. Therefore, by considering Z values, the new BE is associated with Service 2 which is depicted in Table 9 in green.

Suppose other changes include adding a *Calculate Price* EBP and an *Order* BE to our original matrix simultaneously (category 3), deleting the *Add an item* EBP (category 4), deleting the *Warehouse Voucher* BE from the matrix (category 5), and deleting the *Add an item* EBP and *Warehouse Voucher* BP simultaneously (category 6). Z values of these steps are depicted in the fifth to eighth rows of Table 8. Also, the first row of Table 8 illustrates the initial objective function values.

Table 9 Adding a BE to the CRUD Matrix

	Customer	Credit	Accounts Receivable Note	Order	Discounts	Inventory	Warehouse Voucher
Add a customer	C	C					
Add an accounts receivable note	R	U	C				
Receive order	R	R		R			
Receive order	R			C			
Calculate discounts				R	R		
Check inventory				R		R	
Add discounts					C		
Calculate price				R	R		
Add an item						C	
Add a warehouse voucher		R				U	C

4.6 Discussion of Research Challenges

Software evolution processes have a number of well-known challenges, which have been identified and classified by researchers during the last two decades. Every new method in software evolution must contribute to alleviating some of these challenges to decrease the cost of evolution and maintenance. Thus, this section enumerates some of the well-known challenges and clarifies how ASMEM contributes to handle them.

The enumerated challenges in this section are derived from previous research. In (Mens, Wermelinger, et al. 2005, Bass, Clements and Kazman 2003) Mens et al. classify some of the important challenges in software evolution that were identified during a workshop on Challenges on Software Evolution (ChaSE 2005).

- Common software evolution platform

  The challenge is to develop and support a common application framework for doing joint software evolution research. Because Eclipse has the advantage of visibility, industrial acceptance and also reusability of certain components, and the ASOM-Tool was built using the Eclipse platform, this challenge is thoroughly met.

- Support for model evolution

  Evolution techniques should be raised to a higher level of abstraction than higher level artifacts such as analysis and design models, software architectures, requirement specifications, and so on. In this regard, ASMEM localizes the business model changes to the service model, which is a high-level artifact. In fact, business changes mostly and directly affect business process models. Because the service model is based on the business process model, any possible business process model modifications immediately influence the service model. Therefore, the challenge of supporting model evolution is supported by ASMEM.

- Support for co-evolution

  This challenge refers to the need to achieve co-evolution between various types of software artifacts or diverse representations of them. ASMEM propagates changes from the design model (i.e., service model) down to the lower level phases of the life cycle (e.g., specification or realization, among others). Therefore, co-evolution between different types of artifacts or different representations of them is achieved.

- Need for improved predictive models

  This challenge refers to the point that some models are necessary for predicting a variety of things, including where the software evolves, how it will evolve, the effort and time that is required to make a change, etc. In ASMEM, every possible change scenario is identified and handled. Moreover, in ASMEM, every change in the business model can be traced down to the other artifacts, such as the service model.

4.7 Conclusions and Future Work

We have provided a novel method, called ASMEM, which consists of several steps based on quantitative criteria to support automated evolution management in service-based solution life cycles. Because we have adopted a well-defined matrix as the business model, we are able to automatically localize the several types of change scenarios and evaluate incremental changes. In addition, because we have developed automated methods for service modeling, we can automatically propagate the incremental changes to the service model. Moreover, we have developed a tool to implement the service modeling and evolution management methods to supply automation and traceability features to the provided framework. By adopting this framework, changes induced by the maintenance activities in service-oriented systems can be treated automatically, incrementally, and efficiently, thus reducing the cost of maintenance. The method has been verified via a case study.

Future work for the ASOM-Tool can be divided into two main categories: improving the techniques used within the ASOM-Tool, and increasing the ASOM-Tool’s utility by expanding its scope.

References

Arnold, Robert S. *Software Change Impact Analysis.* Los Alamitos: IEEE Computer Society Press, 1996.

Arsanjani, A., S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and H. Holley. "SOMA: A method for developing service-oriented solutions." *IBM Systems Journal* 47, no. 3 (2008): 377-396.

Barr, Michael, and Charles Wells. *Category Theory for Computing Science.* Upper Saddle River: Prentice-Hall, Inc., 1990.

Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* New York: Addison-Wesley, 2003.

Bennett, K. H., and V. T. Rajlich. "Software Maintenance and Evolution: A Roadmap." In *The Future of Software Engineering*, by A. Finkelstein. Limerick: International Conference on Software Engineering, 2000.

Boehm, Barry W., and Phillip N. Papaccio. "Understanding and Controlling Software Costs." *IEEE Transactions on Software Engineering* 14, no. 10 (October 1988): 1462-1477.

Hearnden, D. "Deltaware: Incremental Change Propagation for Automating Software Evolution in the Model-Driven Architecture." *Ph.D. Thesis.* University of Queensland, 2007.

Jamshidi, P., M. Sharifi, and S. Mansour. "To Establish Enterprise Service Model from Enterprise Business Model." *IEEE Computer Society*, 2008: 93-100.

Jamshidi, P., S. Khoshnevis, R. Teimourzadegan, A. Nikravesh, and F. Shams. "Toward automatic transformation of enterprise business model to service model." *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems.* Vancouver, 2009. 70-74.

Jamshidi, P., S. Koshnevis, R. Teimourzadegan, A. Nikravesh, and F. Shams. "An automated method for service specification." *Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010.* Cape Town: International Conference on Software Engineering, 2009. 25-28.

Lehman, Meir M., and Juan F. Ramil. "Software Evolution: Background, Theory, Practice." *Information Processing Letters*, 2003: 33-44.

Mens, Tom, and Tom Tourwe. "A Declarative Evolution Framework for Object-Oriented Design Patterns." *Proceedings of the International Conference on Software Maintenance.* Florence: IEEE Computer Society Press, 2001. 570-579.

Mens, Tom, Michael Wermelinger, Stephane Ducasse, Serge Demeyer, Robert Hirschfeld, and Mehdi Jazayeri. "Challenges in Software Evolution." *International Workshop on the Principles of Software Evolution.* 2005, 2005.

MESOA 2009. *Call for Papers.* September 24, 2009. http://www.sei.cmu.edu/workshops/mesoa/2009/ (accessed February 1, 2010).

Nikravesh, A., et al. "A Tool for Automating Service-Oriented Modeling." *Automated Software Engineering Research Group.* 2009. http://aser.sbu.ac.ir (accessed February 4, 2010).

Shaw, Mary, and Paul Clements. "The Golden Age of Software Architecture." *IEEE Software* 23, no. 2 (2006): 31-39.

Tourwe, Tom, and Tom Mens. "Automated Support for Framework-Based Software Evolution." *Proceedings of the International Conference on Software Maintenance.* Washington: IEEE Computer Society, 2003. 148.

Wahli, Ueli, et al. *Building SOA Solutions Using the Rational SDP.* IBM Redbook, 2007.

Wiels, Virginie, and Steve Easterbrook. "Management of Evolving Specifications using Category Theory." *13th IEEE International Conference on Automated Software Engineering.* Honolulu, 1998. 12.

Yang, Hongji, and Martin Ward. *Successful Evolution of Software Systems.* Norwood: Artech House, Inc., 2003.

5 A Funny Thing Happened on the Way to SOA: Insights from a Three-Year Experience with a Telecom Company

Paulo Rupino da Cunha (CISUC, Department of Informatics Engineering, University of Coimbra, Portugal), Paulo Melo (INESC Coimbra and University of Coimbra, Portugal) and Catarina Ferreira da Silva (CISUC, University of Coimbra, Portugal)

Abstract

We describe three challenges that a major telecom company faced on its way to SOA after the traditional first step of exposing legacy functionality and orchestrating it in higher level business processes. We show how the accepted practice of gradual migration can cause the materialization of a SOA-related "recent-legacy" that may constrain future architecture evolution; we discuss how newly gained agility in changing business processes can have complex consequences in managing the lower level XML involved in interactions with the leveraged "old-legacy"; and, finally, we address an emerging problem of finding the right services to build new business processes in a pool that steadily increases as the migration to SOA progresses. Our solutions are overviewed, and a set of lessons is compiled to raise awareness of potential pitfalls when entering more advanced stages of SOA evolution.

5.1 Introduction and Case Study Background

Competitive markets constantly pressure companies to become more agile, forcing them to leverage their legacy systems in novel ways: new business processes must be designed and quickly deployed to support new products or services; existing ones need to be reengineered or frequently tuned to keep up with best practices in the industry. The need to cope with these constant reconfigurations of existing and new functionality is one of the main drivers for the evolution towards service-oriented architectures (SOA) (Josuttis 2007). However, this migration is complex: IT economics dictate that the investments in existing systems must be preserved, thus discarding greenfield approaches. On the other hand, the risk of causing disruptions in customer service while modifying those systems must be minimized. These constraints usually lead to incremental evolution scenarios. We describe facets of one such project in a major telecommunications company with emphasis on the impact on business processes.

Advances in technology enabled the once independent fixed telephony, Internet access, and TV services to converge into a unified "triple-play" offering supported by IP (Internet Protocol) networks. This move, however, also meant that some of the once independent and heterogeneous systems that supported each of those original services would have to cooperate to enable the new business processes required for provisioning and overall management of the new unified product.

We have been working with the Operations Support Systems (OSS) Department of a telecom company in such a project. This group handles the systems used for service provisioning, including maintaining the network inventory, configuring resources, and monitoring operations. Several internally developed three-tier applications already existed, and the challenge is to migrate to a more flexible architecture that is capable of keeping up with the dynamics of the market in a world of convergence of telecom services.

The initial step was to build adapters that knew how to interact with each specific legacy application and expose required pieces of business logic, as web services, on an as-needed basis. The new business processes were then written as orchestrations of those web services using Business Process Execution Language (BPEL) to promote the required flexibility in changing them or adding new ones.

Having successfully completed this first stage of the paradigm shift from stovepipe applications to SOA, fresh challenges emerged. We will describe three additional steps in the journey of this company:

- the upgrade of SOA components used in the first stage, which revealed that performing incremental migrations introduces a new layer of "recent-legacy" that may constrain future evolution
- the development of a process design-time application to help cope with the complexity of communicating with the legacy systems using XML documents and associated validations and transformations
- the exploration of semantic technologies to assist business analysts and software engineers in dealing with an increasing number of services from diverse sources that are used to compose business processes

The remainder of this paper is organized as follows: we start by delving deeper into these three challenges to raise awareness of these potential pitfalls. We then present an overview of our solutions for each of those situations. In the last section, we talk about the lessons that we have learned so far, just before concluding with some remarks about future work.

5.2 Three Challenges on the Way to SOA

Each of the three challenges is described in a separate sub-section. First, we address the inadvertent introduction of a SOA-related legacy; second, we discuss how modifying high-level business processes may not be as simple as commonly advertised; and finally, we move to our present concern of making sure the business analyst does not become overwhelmed by hundreds of services when trying to design new business processes or reengineer existing ones.

5.2.1 The Unexpected "Recent-Legacy"

The migration to SOA entails dealing with existing legacy systems. However, since that migration is a lengthy process, a second layer of "recent-legacy" may appear: one consisting of SOA tools used early in the project, but that need to be replaced due to technical reasons (such as inability to handle an increasing load) or commercial decisions by the vendors (such as sunsetting of a product). In this case, a rip-and-replace strategy may not be feasible; namely, when the tool is supporting business processes lasting for months or years and the migration of data is highly risky or impossible. In fact, in the telecommunications industry, some business processes have long life spans. Consider, for instance, the case of a contract between a telecommunications company and a property developer, by which the future buyers of the apartments to be built will have special conditions for cable TV and broadband Internet access for a specified period of time. The business process supporting that type of contract is initiated when the building construction is started, but it will terminate a few years later, when the new owners move in and eventually decide to take

advantage of the promotion. If, at some point in time, the telecom company decides to change how such contracts are handled, those already initiated must be honored by the previous rules. However, waiting for all the “running” processes to end before starting the new procedure is unfeasible—it would mean stopping making contracts until the new rules were in place (to avoid constantly pushing forward the end of the last contract) and then waiting for months or years for the last process to conclude. This would defeat the business agility that SOA makes possible. Much like this scenario of modifying long-running business processes, if the infrastructure supporting them needs to be changed, the same problem emerges of allowing those already running processes to complete.

These systems introduced in the first stage of an evolution to SOA, which must be kept running because of established long-running processes, are examples of what we call “recent-legacy.” Our case at the telecom company exhibited this architectural evolution challenge. The existing BPEL engine was becoming unreliable and unable to handle the increasing load, but some of the deployed business processes were long-running and could not be migrated—there were a lot of unknowns regarding how the engine stored its internal state, and some might have pending callbacks from external systems. A solution had to be found to enable a new engine and the old (the “recent-legacy”) to coexist until all processes in the latter concluded. This effectively constrained the path to SOA, forcing the adoption of an intermediate pattern.

5.2.2 Being Flexible Can Be Hard

The ease of modification of business processes is one of the great selling points and promises of SOA. In practice, at a high-level, this procedure often entails modifying invoked services or adding new parameters to be handled. At a lower level, however, these changes may ripple throughout all communication between the process and its building blocks—the services exposed from the stovepipes. In our case, an initial XML document was supplied to the business process and used to collect information as it was passed along the various services involved in provisioning a telecom product. Changing the data contained or expected in the initial XML meant that validations at every service entry (using XML Schema Definitions (XSD) (van der Vlist 2002)) and all required modifications throughout the process path (using Extensible Stylesheet Language Transformations (XSLT) (Clark 1999)) would have to be revised, and these many documents changed accordingly to account for that new piece of data. Manually changing these XSD and XSLT documents is an arduous and repetitive task, and is also prone to errors that are hard to detect until the process is in operation. If the flexibility in high-level business process change is to be attained, such tasks must be supported by technology that prevents (or at least flags) the errors and decreases the overall complexity of making changes.

5.2.3 Services Everywhere: How to Find the Suitable One?

Currently, we face a third challenge stemming from the rise in the number of available services, caused by the normal evolution towards SOA. Although registries or repositories are frequently pointed out as the solution to list which services are available and how to invoke them, when composing new processes, this syntax-oriented mechanism is of decreasing usefulness. In this situation, the analyst needs to be able to query the existing pool of services in terms of semantics to find perfect or close matches to desired business logic, regardless of the actual service names. So, more than just list the services and how to invoke them, we need information about their meaning in the business context. It is not feasible to sift through hundreds of services, reading

each description every time a new business process needs to be created. A mere name-oriented (syntactic) search is not reliable due to inconsistencies and biases towards engineering terms.

5.3 Addressing the Challenges

Following the same format used in Section 5.2, we now present an overview of our solutions to each of the three challenges we previously described. First we show how we mitigated the effects of the “recent-legacy,” then we show how automation was used to lessen the low-level effects of changes in business processes, and finally we explain our current strategy of adding semantics to the emerging SOA. The first couple of solutions were implemented in two separate projects that totaled an effort of eight person-years, and included the development of two systems comprising around 50,000 lines of Java code. The system for addressing the third challenge is still under development. Five autonomous telecom “legacy” systems were involved in these projects.

5.3.1 Versioning—Outside the Box

To cope with the “recent legacy” problem, we created a self-contained component that implements the versioning usually required by long-running processes. Although some BPEL engines support this feature internally, we needed to manage this as an independent capability, such that we could seamlessly handle both the old and new BPEL engines. Additionally, callbacks from external services also needed to be routed to the appropriate engine in a transparent fashion. Using this approach, processes already deployed to the engine being retired can still be accessed until they terminate, but new versions are created in its new replacement.

Versioning must be transparent for all involved. Business analysts should not be concerned with how IT ensures co-existence of “old” and “new” versions of business processes. External users of the processes should not have to modify their systems every time a new version is deployed. As a result, deployments and invocations should only need the original process name, regardless of how many of its versions are already running and how many BPEL engines are in operation. The versioning system must thus ensure that new deployments do not overwrite existing (older) ones, and that requests to business processes are always forwarded to the correct version in the correct engine.

Our component consists of two modules: the *Deployer* and the *Gateway*. The *Deployer* intercepts deployments from the business process editor and modifies the data to add and track a version number. The *Gateway* intercepts all external communication with the business processes to ensure that it is always forwarded to the correct engine and version. In both cases, the modules act transparently to the involved parties. All the data about the BPEL processes under version control is kept in a database, to ensure persistence in case of planned or unplanned reboots.

As Figure 10 shows, the original BPEL process sent by the editor is intercepted by the *Deployer*, which determines the next version and modifies the name accordingly before letting the transaction proceed. The answer returned by the engine is then passed back to the editor, whose user is unaware of this intermediate processing. Also, from the point of view of the engine, all this occurs as if just another independent BPEL process had been deployed. The information about the mapping of the original file (in the editor) to the multiple file versions (inside the engines) resides in the versioning system only. To conclude, the *Deployer* notifies the *Gateway* to refresh its information about the processes currently deployed.

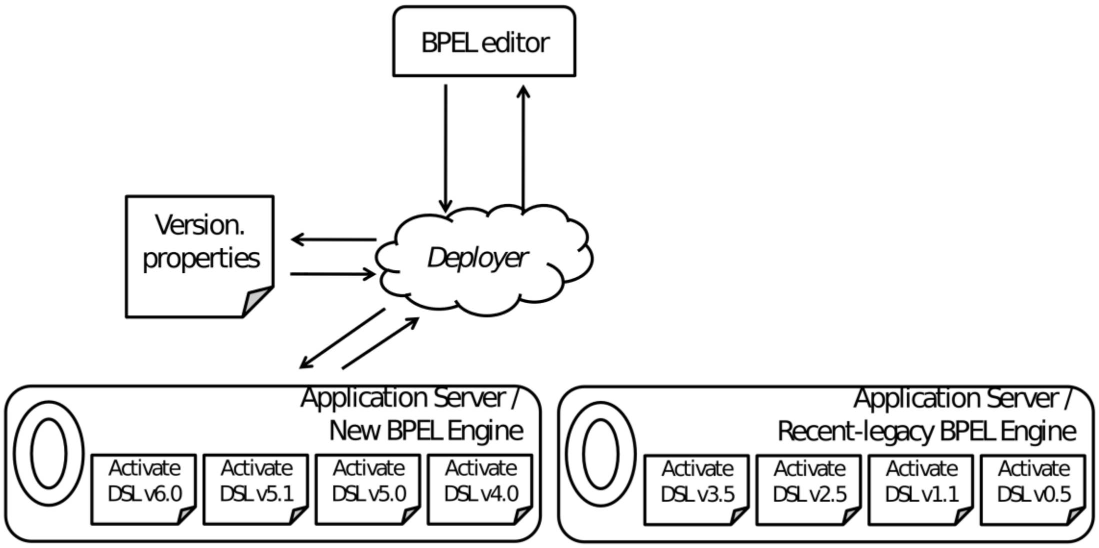

Figure 10 Intercepting the Process Deployment

From the moment a business process is installed in a BPEL engine, it becomes available for instantiation and interaction with other systems. As in previous versions, the client only needs to know the original process name, and the *Gateway* module will take care of all necessary intermediation to ensure communication is done with the correct version, in the correct engine, as shown in Figure 11.

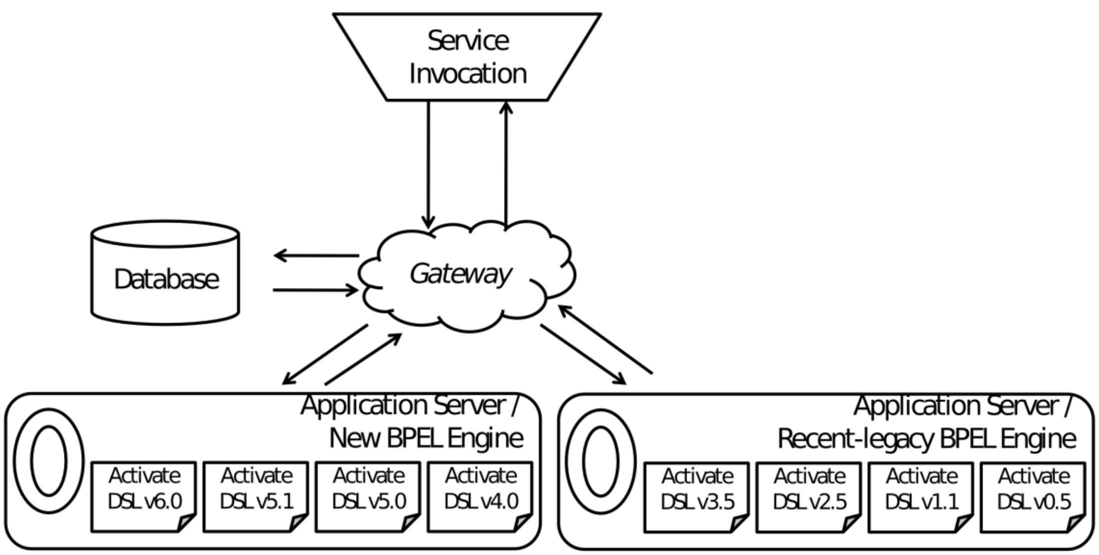

Figure 11 Forwarding an External Invocation to the Correct Process Version

The *Gateway* works by intercepting invocations to the processes, finding out for which version any given process is intended and in which engine it resides, calling it, and then returning the answer to the original caller. The complexity resides in ensuring that messages that arrive in response to an interaction initiated by an older version of a process are delivered to that specific instance and version, rather than to the most current one. This is achieved by tracking and manag-

ing BPEL's correlation IDs, which are frequently used, and sometimes required, in asynchronous calls.

5.3.2 A Little Data Orientation in a Control-Oriented World

The general problem of maintaining the validation (XSD) and transformation (XSLT) documents required to communicate with the web services exposed from the legacy applications can be a hard one (since it must include any kind of processing required to create the actual mappings), but can be made a little easier by observing the patterns of data usage on the processes involving them. Although BPEL is a control-oriented language, the processes in our case were actually data-oriented: the flow was designed to integrate data provided by the various operations support systems as a process advanced from one step to the next towards completion.

The scenario we found was that the process orchestrator sequentially invoked each stovepipe adapter web service, passing it the information it needed and receiving in turn information containing the changes it had made or eventual "new" information, as shown in Figure 12.

In such a scenario, only a subset of the capabilities of the XSDs and XSLTs is required. With this simplification, it becomes feasible to automatically generate or validate these documents every time the interactions with the legacy adapters are changed.

To create a system that allowed such generation, we had to stop thinking of the processes as control flows of services and start viewing them as data flows.

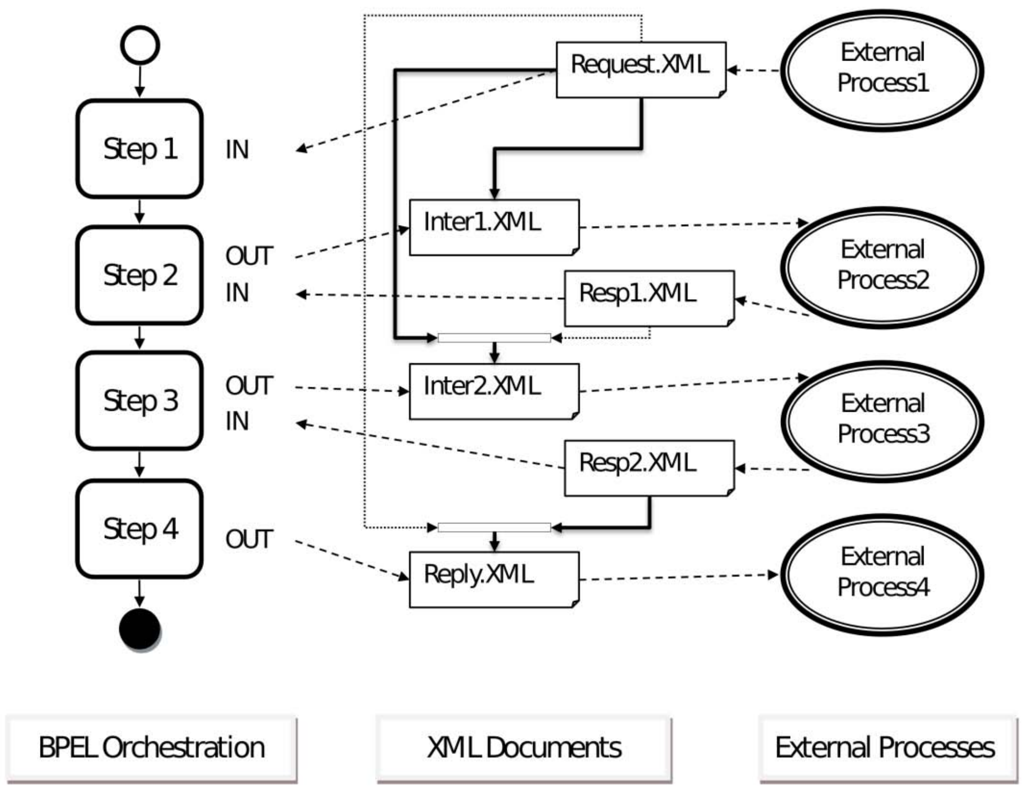

Figure 12 Process Structure Describing Adapters Orchestration

We built an application that allowed us to describe how each data element was modified by different operations (services) as it "traveled" from the beginning to the end of the business process. We accounted for multiple possible origins and also for the fact that paths were not linear, since the same data element could be delivered to different legacy systems. Finally, since the same data element could be used in different ways in each stovepipe application, we had to allow it to be named (and placed in the XML tree) according to the invoked service or step and according to the constraints or default values required by that step.

Once the decision was made to focus on the data flow for each component, a system to help describe those components could be built. That system could in turn be used to generate the required XSD and XSLT documents. This design-time system has a Graphical User Interface (GUI) that allows the definition of the data path for each information component and the aggregation of simple data in structures on each application invocation.

However, even with the system in place, it was obvious that even for processes with the simple structure that was required, inputting the information describing the data processing in each step required significant domain knowledge. This included knowledge on which services are present and on the data used by the stovepipe applications exposed via adapters (which data and in which format). Rather than forcing common semantics, most often the adapters just exposed the internal data and process mechanisms used by the stovepipe application. We tried to lessen the burden by providing capabilities to import previously defined templates, and therefore help to reuse previously codified knowledge about the data.

5.3.3 Enhance Web Services Visibility with Semantic Technologies

To not overwhelm a business analyst with a huge number of available web services, which are usually developed and described using the WSDL, Web Services Description Language (Chinnici, et al. 2007) by different teams, a semantic search—one that goes beyond the mere syntax and into the business meaning—is in order. This can be achieved with appropriate annotations that link concepts in service descriptions to ontological concepts (Guarino 1998).

The solution we are developing explores this avenue by introducing a semantic registry of services, which are described using specific languages, such as WSDL and Semantic Markup for Web Services, OWL-S (Martin, Burstein, et al. 2004), as shown in Figure 13. OWL-S proposes a particular semantic framework within which to characterize the semantics of Web services, by specifying language constructors for *service profile*, *service model*—both to characterize the service—and *service grounding* to bind it, i.e., how to interact with the service. It adds some attributes to WSDL extensions in order to connect both languages. For example, OWL-S specifies parameters that map to WSDL message parts, such as *operation*, *port type*, *binding*, and *service* constructs. Furthermore, Martin et al. suggest that a WSDL operation can refer to an OWL-S atomic or composite process and a WSDL interface should refer to an instance of a profile class (Martin, Paolucci and Wagner 2007)

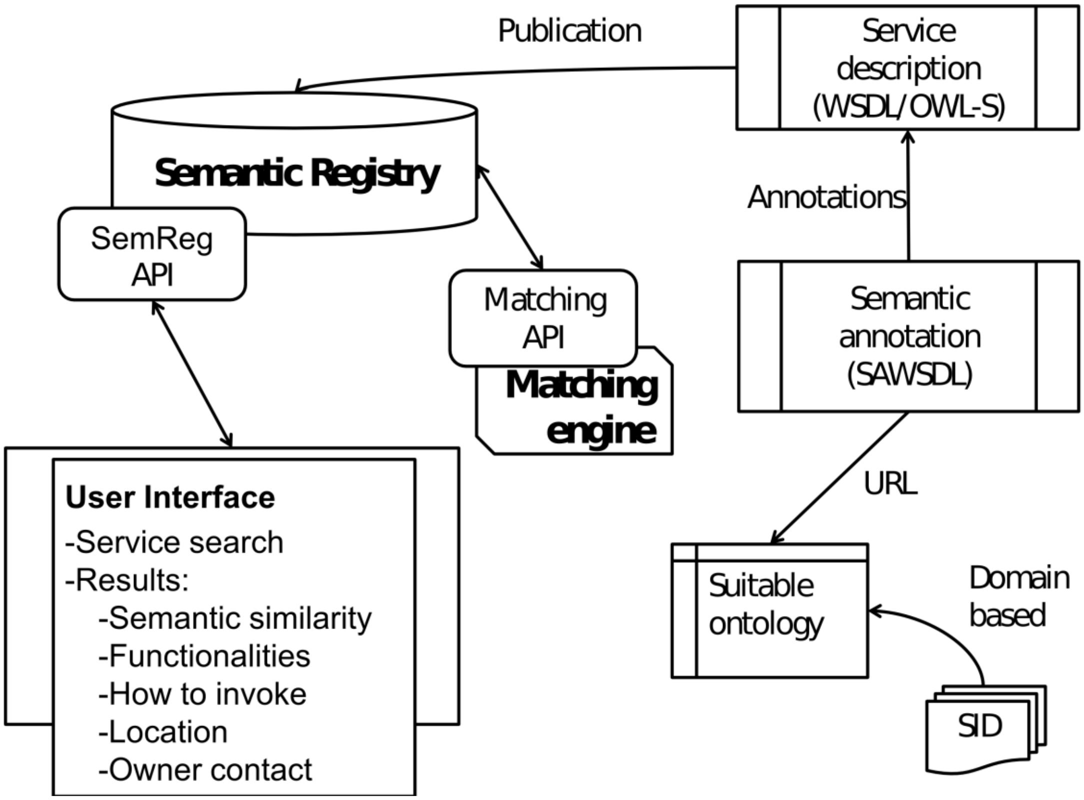

Figure 13 Semantic Registry Scenario

Specific SAWSDL (Semantic Annotations for WSDL and XML Schema) attributes (Farrell and Lausen 2007), such as MODELREFERENCE, enable the annotation of service descriptions, providing links to explicit concepts that are defined in a suitable ontology. Since none exists at the moment for the telecom domain, we are building a draft from other accepted standards in this world, such as the Shared Information/Data (SID) model (Reilly and Wilmes 2008) from the TeleManagement Forum.

Some experimental semantic registries already exist under open source licenses (Kourtesis and Paraskakis 2008), as do matching engines that can compare ontological concept definitions referenced by SAWSDL MODELREFERENCE attributes. OWL-S-specific engines can also detect similarities between services defined in this language.

Assembling these components together with some additional logic, we can provide the business analyst with a Google-like user interface to a semantic search that takes a business-level description of what he or she needs and returns the most probable matches available in the registry sorted by similarity. Additional details may include service functionality, invocation information, location, and owner contact. This possibility of querying the available services in terms of business concepts is very valuable to find the right building blocks when designing or reengineering processes.

5.4 Some Lessons, So Far

In our work with the telecom company, we have learned several lessons regarding the evolution to SOA. A few of them are not new, but are worth restating. Surprisingly, we also found out that following some accepted practices may lead to new problems.

Moving to SOA frequently involves understanding and adapting existing “stovepipe” applications to expose some services. Their complex business logic would be hard to duplicate and difficult to debug if implemented “from scratch.” To do so would also require infeasible time spans, large investments, and great risk.

Risk is also present in “service-enabling” these current mission-critical systems, so the transition to a future SOA should be gradual. Mistakes in this step may potentially affect a large number of customers. Unexpectedly, this incremental evolution may create challenges of its own. In fact, in our case, a BPEL engine had to be replaced due to scalability and reliability problems, but running processes could not be migrated to the new product due to their nature, as they were long-running orchestrations of services provided by external systems. A SOA-related “recent-legacy” had been introduced by the gradual migration, and was raising different challenges than the ones presented by the traditional “stovepipes.” Eventually, a “gateway” had to be developed to allow old and new BPEL engines to work in parallel for a few years, until all processes running in the former complete their execution.

Although not exclusively present in SOA migrations, a second situation, involving the sunsetting of a different product by the vendor, indicated the need for tighter control over the evolution of key systems in the company’s portfolio—including those required by the SOA technologies used. Strong-copyleft licenses, like the GNU General Public License (Stallman 2007) fill this requirement better than less-rigid ones. For example, the overall solution proposed for the company included a GPL-licensed BPEL engine and an editor that was simply free-of-charge under vendor terms. After a couple of years, the GPL-licensed product is still available, but the freeware component has become a commercial product. In spite of its predictability, however, use of GPL code requires a strong discipline. Care must be taken to isolate its components in a way that the fast evolution of the open-source world doesn’t keep forcing the company into constant upgrades. Such isolation also helps in delimiting which parts of the company’s code base must be shared with the community, according to the license terms.

The process of “service-enabling” the legacy also provided some learning points. Usually, the easiest way to expose functionality of each existing application is to create a “fat” adapter that allows access to its full capabilities. However, while such solutions may allow for faster integration of the application in new “SOA enabled” processes, it also makes those processes harder to build, since those adapters tend to do more than just expose core functionality: they often also expose the original application’s internal structure and logic. This can happen, for example, by requiring the invocation of capabilities to be done in a certain order, or by forcing it to include additional information, or to have its data presented in a particular way. Moreover, when integrating different stovepipes whose services have been exposed in this way, one usually discovers that the logic and requirements of different applications are extremely diverse, preventing common solutions. This increases both the cost of creating and maintaining the adapters, and also thesharing of information among the different applications.

Finally, using the services originating from several vertical legacy systems in transversal, end-to-end, long-running processes, may raise a new problem: the need for rollbacks or cancelations. In a typical telecom provisioning process, for instance, several network resources have to be reserved in order: physical circuits, logical configurations, field teams. These are usually handled by different legacy applications. If one of these resources is unavailable, then the ones already acquired must be relinquished. This can be hard to implement, since it is not usually supported by the original legacy systems.

5.5 Future Work

The introduction of the semantic registry/repository poses new socio-technical challenges. There are normal governance considerations, such as the attribution of decision rights to add, change, and retire services, but also specific issues regarding how the services are semantically described and by whom. The reference ontology has to be agreed on and maintained. Decisions have to be made on whether developers become responsible for annotating their services, or if a dedicated team should be constituted. Resistance to change may arise, so reward mechanisms should be discussed in tandem with the technological solutions to ensure effectiveness.

On a different level, although our present experiments focus on the use of the semantic registry/repository at process design-time, its use for runtime dynamic discovery and binding is being considered in the longer term. In that case, selecting from multiple matches returned by a service search is not trivial. A rules mechanism must be able to account for both technical and business concerns, such as simultaneous interoperation of interdependent services, quality of service, and cost. Processing performance of this kind of computation will also be an issue.

5.6 Acknowledgements

The authors are indebted to the reviewers for their comprehensive comments and suggestions.

References

Chinnici, R., J.-J. Moreau, A. Ryman, and S. Weerawarana. "Web Services Description Language (WSDL) Version 2.0, Part 1." *W3C Recommendation.* June 26, 2007. http://www.w3.org/TR/wsdl20 (accessed February 5, 2010).

Clark, J. "XSL Transformations (XLST) Version 1.0." *W3C Recommendation.* November 16, 1999. http://www.w3.org/TR/xslt (accessed February 5, 2010).

Farrell, Joel, and Holger Lausen. "Semantic Annotations for WSDL and XML Schema." *W3C Recommendations.* August 28, 2007. http://www.w3.org/TR/sawsdl/ (accessed February 5, 2010).

Guarino, N. *Formal Ontology in Information Systems.* Netherlands: IOS Press, 1998.

Josuttis, N. *SOA in Practice.* Sebastopol: O'Reilly Media, 2007.

Kourtesis, Dimitrios, and Iraklis Paraskakis. "Web Service Discovery in the FUSION Semantic Registry." *Lecture Notes in Business Information Processing.* Innsbruck: Springer Berlin Heidelberg, 2008. 285-296.

Martin, David, et al. "OWL-S: Semantic Markup for Web Services." *W3C Recommendations.* November 22, 2004. http:///www.w3.org/Submission/OWL-S/ (accessed February 5, 2010).

Martin, David, Massimo Paolucci, and Matthias Wagner. "Towards Semantic Annotations of Web Services: OWL-S from the SAWSDL Perspective." *OWL-S Experiences and Future Developments Workshop.* Innsbruck: European Semantic Systems Initiative, 2007.

Reilly, John, and John Wilmes. *Application Integration Using the SID.* Morristown: TMForum, 2008.

Stallman, R. *The GNU General Public License.* June 29, 2007. http://www.gnu.org/licenses/gpl.html (accessed February 5, 2010).

van der Vlist, E. *XML Schema.* Sebastopol: O'Reilly Media, Inc., 2002.

6 Towards a Design Approach for an Effective System Evolution of a Large Electronic Archive Information System

Quyen L. Nguyen (U.S. National Archives and Records Administration, USA)

Abstract

Providing web-based access to a long-term digital preservation system is a challenging endeavor. The amount of digital-born materials is huge, due to the fast pace of information technology and its widespread utilization in government, business corporations, and academic institutions. On top of the volume challenge is the extremely varied nature of the data, ranging from office automation, geospatial images, to multimedia artifacts. The core issue discussed in this paper stems from the current goal to make digital objects accessible via the web, such that the access can be widespread and independent of platforms and software applications used to create these objects. Consequently, a Web-Based Archive Information System (WAIS) will have to face a double system evolution. As with any software system, WAIS has to cope with normal evolution over time. More importantly, WAIS must be able to adapt to the evolution of the very software applications that generated the digital objects. This paper will study the concept of software evolution as it is applied to WAIS. From that study, we will analyze a multi-level approach from system architecture to software and data design. We will show that this approach, which includes a recursive scheme for service decomposition, should contribute to an efficient system evolution and maintenance for the SOA-based WAIS in order to cope with its inherent evolution characteristics.

6.1 Introduction

The main goal of an electronic archive system is to provide long-term preservation and access to electronic records in such a way that it is independent of the hardware and software that created them. For an archive system to store, preserve, and provide search and access to electronic records, it must overcome the following challenges.

- **Variety.** In the case of a federal or state government, the system must deal with different application domains, which may include health care, education, defense, space exploration and energy, or environmental protection. As a result, records will contain various types of knowledge. Moreover, since they were created by various applications, their manifestation and representation will have different formats, which may include Microsoft Office documents, relational database files, geospatial images, or multimedia materials.
- **Obsolescence.** By the time the digital objects are ingested into the system, the creating applications and platforms will most likely either be obsolete or belong to old versions of the software and hardware.
- **Volume.** It is estimated that the total volume of incoming records will be enormous and will continue to grow over the years. Petabyte and exabyte ranges of data are not unimaginable.

In addition to these business challenges, the system architecture should be designed in such a way to satisfy the non-functional requirements of extensibility, scalability, security, and user-friendliness. As part of the latter, the current system must be web-based. On one hand, it will be

able to reach out to as wide a public as possible. On the other hand, by delivering the content via the ubiquitous web browser, we will not have to levy high and complex system requirements to record researchers. This will create a challenge in its own right as shown later in this paper.

The remainder of the paper is organized as follows. In section 6.2, we study the software evolution concept, analyze the properties of WAIS, and discuss how the evolution concept applies to WAIS. Section 6.3 summarizes some related work. In Sections 6.4 and 6.5, we propose and analyze a multi-level approach for WAIS to respond to the software evolution laws discussed in Section 6.2. At a high level, this approach is based on the Open Archive Information System (OAIS) reference model (Consultative Committee for Data Space Systems 2002), and exploits current software engineering methodologies, notably service-oriented architecture (SOA). The paper ends with a conclusion and summary in Section 6.6. The contributions of the paper are: a) to introduce the concept of evolution relativity that is inherent to a system like WAIS; b) to present a data-centric view of software evolution within the context of WAIS; c) to propose a practical and effective approach from the perspectives of system evolution and maintenance and within the context of the digital preservation domain and the SOA framework.

6.2 System Evolution Characteristics

As with any software systems, the designers of WAIS must consider the evolution that is bound to happen locally within the system itself.

6.2.1 Local Evolution

According to (Perry 1994), the factors that enable system evolution are:

- **Domain**, which represents the real-world application that the software system is trying to model.
- **Experience**, which is gained by the usage of the system over a period of time.
- **Process**, which consists of the methods and technologies used by organizations to realize the system.

The evolution of WAIS is manifested in all three areas of domain, experience, and process.

With respect to domain, WAIS follows the Open Archive Information System (OAIS) reference model (Consultative Committee for Data Space Systems 2002) as shown in Figure 14, which specifies six major components: Ingest, Data Management, Preservation, Archival Storage, Access, and Administration. The input to the system will be a special information package, which contains digital objects. Then, pre-processed data and their metadata will be stored and preserved in Archival Storage in the form of an AIP (Archival Information Package). Output in the form of a DIP (Dissemination Information Package) will be sent out for a consumer requesting access to data in Archival Storage. The access transaction normally starts with a search and discovery.

Potential changes of this model can be found in every component. For example, on the Ingest side, increase in data volume will require higher scalability and throughput. Business rules used to govern data ingest and preservation may change to comply with new regulations. The notion of a "digital record" and essential characteristics to ensure its archival authenticity can evolve (The InterPARES Project n.d.), and impact the Preservation Planning module.

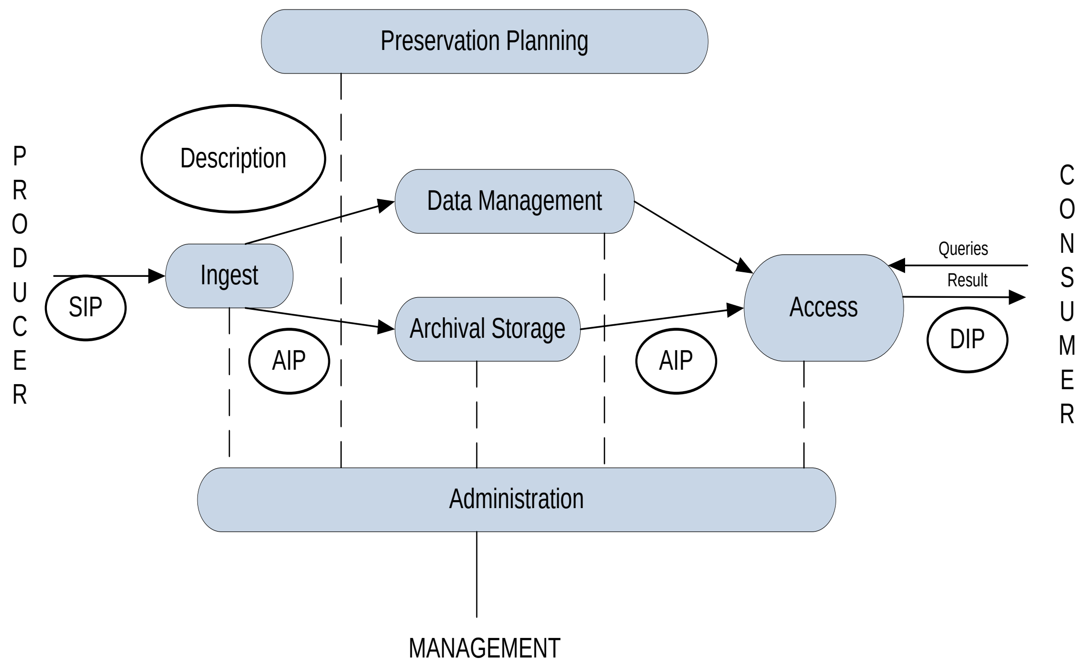

Figure 14 OAIS Reference Model (Consultative Committee for Data Space Systems 2002)

With respect to experience, the users may demand enhanced search capability which must go beyond the well-known keyword search. While search is currently focused on textual documents or metadata of multimedia objects, future demand may be to search a video or still images based on a multimedia clip or figure pattern. As today's users demand more participation when accessing content, features of Web 2.0 should be considered such as social tagging, bookmarking, discussion forums, and interest groups. Another aspect is that the system should be able to evolve to support various access devices. Indeed, the web browser serves as a window to WAIS content, but it can be run on a traditional desktop or a cell phone. Furthermore, a student may want to research historical materials for a school project via her Wi-Fi-enabled handheld game console. Since WAIS is potentially a keeper of vast and diverse content, evolving to the Web 2.0 model may be beneficial and interesting. On the ingest side, agencies becoming more familiar with the system will transfer more records to WAIS, creating the scalability challenge, and therefore the system must be able to adapt to the growth of record volume and the user community.

With respect to Process, software techniques and methodologies used to implement WAIS may change, prompting a system evolution to exploit the latest technologies. For instance, transactional data are currently kept in relational databases, while metadata are kept in XML-based repositories. This approach may change if a new database management model arrives and proves to more adequately fulfill WAIS requirements. This is one of the many examples of Evolvability challenges. Indeed, the system must accommodate new technologies in software and hardware to be inserted using standard APIs and interfaces.

6.2.2 Evolution Relativity

A deeper look at WAIS reveals its special characteristic of system evolution timelines, as shown in Figure 15. First, we have the system evolution intrinsic to any software system; but it is precisely this very nature of software which creates another evolution stream that WAIS has to ac-

commodate. In fact, software applications that created digital objects to be stored by WAIS also evolve, in terms of document formats and behaviors. Meanwhile, the responsibility of WAIS is to preserve these essential characteristics. Let Tc, Ta, and Ts be the times when data are created, archived, and accessed, respectively. The challenge for WAIS consists of transforming itself to utilize the latest technologies of epoch Ta in order to provide long-term access at time Ts (Ts ≥ Ta) to data materials born out of technologies of epoch Tc (Tc ≤ Ta). In some sense, one may say that WAIS looks like a time-traveling "DeLorean," which provides users a view "back to the future" of past digital objects. Thus, there are two timelines of evolution in WAIS: one for the system itself and the other for the incoming digital objects. The latter is always lagging relative to the former.

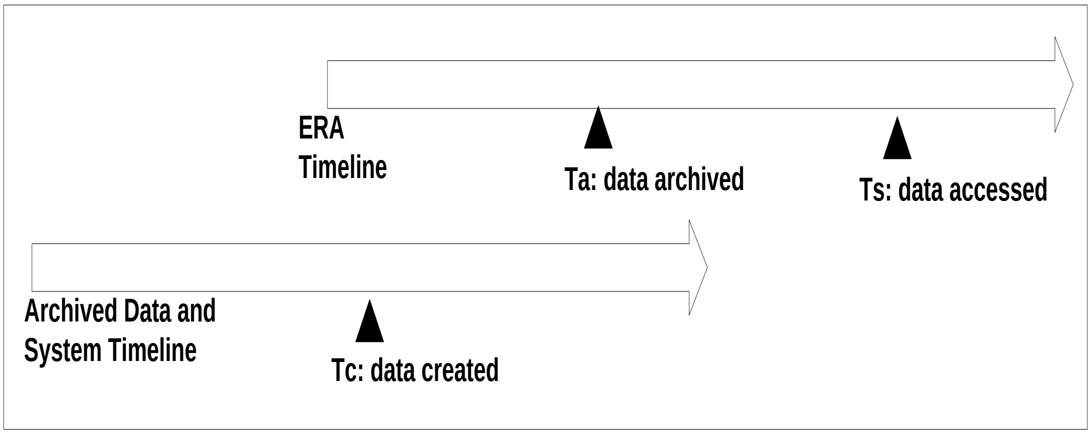

Figure 15 Relative Evolution Timelines

Due to these dual timelines, WAIS has a double challenge in terms of evolution: a normal software evolution and an archival evolution.

6.2.3 Data-Centric System Evolution

The above *Evolution Relativity* is due to the evolving digital objects to be ingested into WAIS. This leads to the need of considering the data-centric dimension of the system evolution. It may be possible to argue that this Data dimension falls under Domain, since Data is part of the application domain of digital archiving. However, in the case of WAIS at least, we believe that eliciting the Data aspect will shed more light into the evolution nature of such a system. In fact, most of the software applications have to deal with data involving current transactions. Moreover, these data are generated in an active environment, with current technology and up-to-date business rules. For WAIS, there are three kinds of data:

1. **Ingest data**, which are usually created by evolved systems, which are at most as current as WAIS, according to the previous section. Note that to-be-archived data are no longer active records—that is, they are not used to conduct day-to-day business operations.
2. **Stored data**, which are currently inside the Archival Storage of WAIS. Since the goal of WAIS is to provide access to those data using current technology, some actions such as data transformation and migration have to be performed, depending on the preservation strategy.

3. **Business data**, which are created by WAIS itself to govern the operations inside the system. These data could be characterized as transactional, and classified as regular enterprise business data.

WAIS has to evolve to support all these categories of data. But, being able to support the newly incoming data stream and preserve the resident data is of particular interest for WAIS. In (Thibodeau 2002), Thibodeau discussed different data preservation strategies, ranging from emulation, migration, persistent format transformation (PFT), to Universal Virtual Computer (UVC). Noteworthy is the fact that these strategies are evolving, and any WAIS implementing any set of them will be bound to system evolution. Moreover, until now, there was no clear winning strategy. The trending thought is that an overarching system will have to support all strategies, each of which may be suitable to certain data types and applications.

On the other hand, the system has to adapt to the *Extensibility* challenge so that new record types, data types, and services could be added without requiring extensive redesign. Research has already been done for preserving data types other than textual documents. Bonardi et al. (Bonardi and Barthelemy 2008) focused on the specific challenges in applying the different preservation strategies above to electronic modules of musical pieces. Kia Ng et al. (Ng, et al. 2008) looked at ways to describe and classify ontologically digital objects containing “Interactive Multimedia Performances” in order to support the preservation and access to those objects.

6.3 Related Work

The issues of architecture reuse in SOA environments have been studied in (Street and Gomaa 2008). Those issues encompass architectural constructs and patterns, service interface, service logic, service messaging, service discovery, and service differentiation. While analyzing each issue, the authors have proposed concrete guidance in order to maximize the reusability of a SOA system. The paper also argued that a common data model is necessary, and that its benefits from the reuse perspective outweigh the performance impact.

In (Saul and Klett 2008), the authors proposed a SOA-based framework to model a digital preservation system. With the guiding principles of scalability, adaptability, and service-orientation, the approach combines two standard models of Service Component Architecture (SCA) and Business Process Execution Language (BPEL) to design services and components within the system. While the first model offers a structural view, the second provides a behavioral aspect of the system.

The aspect of granularity in service design has been discussed in (Papazoglou and van den Heuvel 2006), which suggested leaning towards coarse-grained and high-level services so that they are more useful to business applications.

In our study, we also investigate the granularity issue of service and component modeling in order to not only optimize evolvability, but also to minimize the maintenance cost of the system, due to the peculiar and inherent evolution characteristics that a WAIS system exhibits. Furthermore, our scheme offers a practical and concrete mechanism to determine the appropriate level of service granularity. We will show that the scheme is at least applicable to WAIS, or to any SOA-based digital preservation system.

6.4 Multi-Level Approach

Given the nature of double-system evolution of WAIS, the question is how to make it evolvable to ensure its long life. We have to find an approach to control system changes in order to maximize reusability and minimize the cost and complexity of maintenance and evolution, while still ensuring stability in terms of working system and architecture design. The proposed approach is multi-level, from architecture to software to the data modeling level.

6.4.1 System Architecture

At the system architecture level, the SOA paradigm can be adopted. Indeed, the OAIS reference model lends itself naturally to a SOA architecture, where each component of the OAIS becomes a service in WAIS, as depicted in Figure 16. We have modeled the main services: Ingest, Access, Preservation, and Data Management, which contains Storage, and Administration. The communication backbone between these services is an Enterprise Service Bus (ESB), which also allows the system to communicate with existing legacy systems in the enterprise. The main data object exchanged between the services via the ESB is represented by the AIP, which contains both data and metadata.

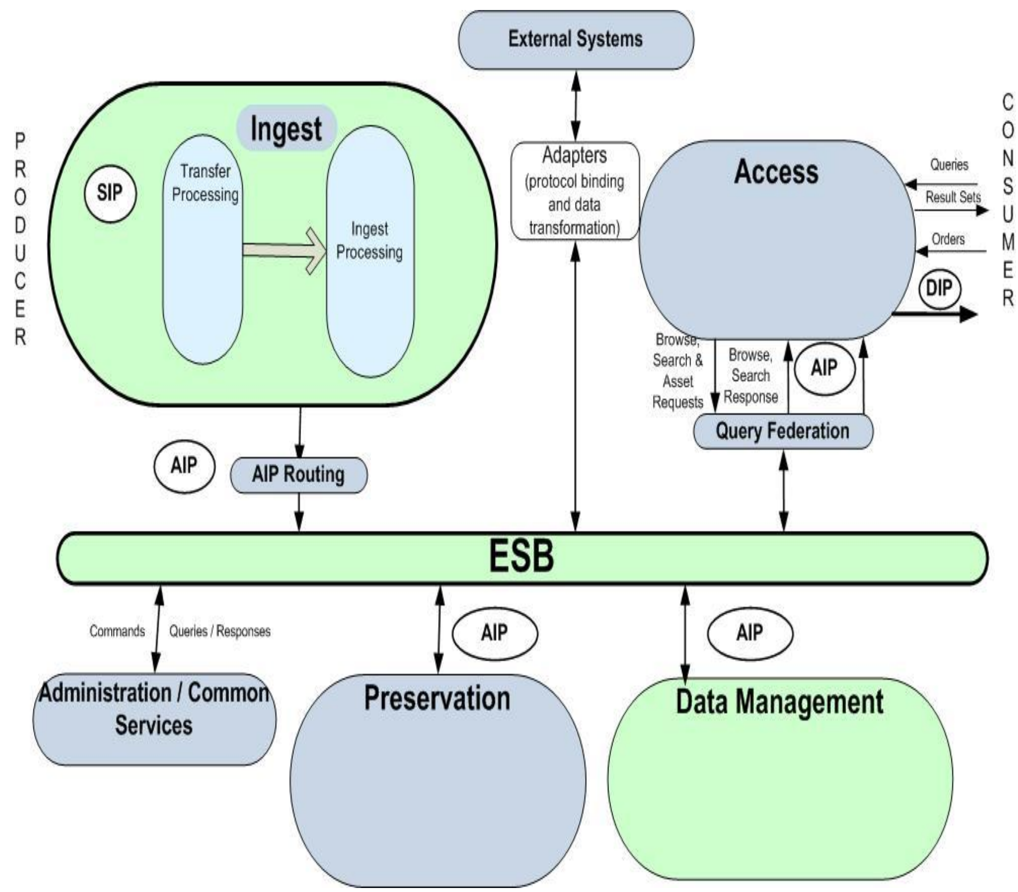

Figure 16 SOA Design for the OAIS System

The quality attributes of a SOA system can be found in the literature. Of particular interest is the evolvability as discussed in (O'Brien, Bass and Merson 2007), which allows adding, and updating services thanks to their loose coupling. For example, when applied to WAIS, the Ingest service

can be easily extended to accommodate new data types. Yet, the interface with other services will still be done via the exchange of AIP. Moreover, components can be developed within each service. These components can be services themselves, and can be chained together by BPEL (OASIS 2007).

Finally, the guiding principle of the architecture is to adopt open standards in protocols, communication, software development, and data management. This is to ensure the interoperability and independence of the services, thus facilitating system evolvability.

6.4.2 Service Decomposition

6.4.2.1 Algorithm

With a careful design of component-services, various workflows and orchestrations can be developed quickly. In the case of preservation, the use of BPEL allows flexible accommodation of preservation methods depending on the level of service and value of the digital records as determined by a Preservation Planner. The key thing in the design is to know the granularity level of a service. That is, we need a scheme that could provide to the software architect a criterion for when to stop in the service decomposition. The generalized design steps of our Service Decomposition Scheme (SDS) are summarized in Figure 17.

```
1   Start with the set of high-level services S = {Ingest, Storage,
2   Access, Data Management, Preservation Planning, Administration}.
3   Call decompose (S).
4
5   Decompose(X):
6   For each service s in X, do:
7       If s=<<tool>>, stop.
8       Else
9           Decompose s into a set of components R.
10          Call decompose(R).
11      Endif.
12  Endfor.
```

Figure 17 Service Decomposition Scheme (SDS)

Note that SDS follows a recursive algorithm pattern, with the starting set containing all services defined by OAIS reference model, and the termination condition expressed in Line 7 above. In our case, the stopping criterion is <<tool>>, which corresponds to the lowest level of the service in the decomposition design process.

Then, what constitutes the stereotype <<tool>>? A service is considered to be a “tool” if there is an implementation provided by commercial-off-the-shelf software (COTS) or free open source software (FOSS). In the domain of digital archiving and preservation, there are software packages available to perform file format identification such as DROID (Digital Record Object Identification) (Geeknet, Inc. 1999), JHOVE (JSTOR/Harvard Object Validation Environment) (JHOVE 2009), and Archival Processing Tool (APT) (Underwood, Hayslett-Keck and Laib 2005). For format transformation, instances of <<tool>> that currently exist are Xena Digital Preservation Software (National Archives of Australia 2009), and Oracle Outside In (Oracle 2009). We will show later how we can apply SDS and make use of those tools in the Ingest and Preservation services.

The pros and cons of the granularity of a service were discussed in terms of governance, overhead impact and system management by Arsanjani (A. Arsanjani 2004) and Papazoglou (Papazoglou and van den Heuvel 2006). In our case, the concept of <<tool>> is bringing us the following benefits: a) to perform service modeling in a deterministic way; b) to establish a framework or platform comprising of to-be-developed components; c) to identify possible ready-made components to be plugged into the framework over time. As we will see in the next section, these low-level components can be wrapped as services to participate in a service composition or orchestration, which can be realized by BPEL.

From the practical point of view, the impact of using SDS is that the design of service decomposition has to be coupled with the activity of researching the software market to find the so called ready-made components. This research activity should follow a standardized process consisting of the following phases: 1) perform market research based on the functionalities required by a <<tool>> that has been identified via the service decomposition exercise; 2) determine a set of criteria and a quantitative scoring system; 3) identify viable candidates for further in-depth evaluation using the available literature and support community such as vendor communication or public developer forums; 4) develop prototypes utilizing a narrowed set of runner-up candidates. During the selection process, the evaluation criteria will contain the non-functional requirements of extensibility, scalability, security, and user-friendliness as listed in the introduction of this paper. Open standards compliance is also an important factor for <<tool>> product selection.

Thus, in order to maintain and evolve our system designed with SDS, we need to monitor the software arena in the areas of tools which can be utilized for the different phases of digital preservation. One can argue that for SDS to depend on a <<tool>> is a disadvantage. Actually, in the case where there is no <<tool>> that can fulfill the functionality, our design can be viewed as a platform with a kind of placeholder for future realization.

6.4.2.2 Example

In order to illustrate SDS, we will apply the scheme to the Ingest service. Similar steps can be derived for other services as well. Figure 18 lists the iteration steps in the decomposition of the Ingest service down to the level of pluggable tools.

```
decompose(Ingest):
Ingest is not a <<tool>>.
Decompose Ingest into a set of components R = {FormatIdentify,
ExtractMetadata, BuildAIP, ValidateIngestData}.
Call decompose (FormatIdentify).
FormatIdentify is not a <<tool>.
Decompose FormatIdentify into a set of components R={DROID, JHOVE, APT}.
DROID is a <<tool>>, stop.
JHOVE is a <<tool>>, stop.
APT is a <<tool>>, stop.
Call decompose (ExtractMetadata).
…
…
```

Figure 18 Decomposition of the Ingest Service

6.4.3 Data Management Design

In the area of data management, most of the objects in the system such as business objects or metadata are represented in XML. The rationale of choosing XML is its open standard and flexibility. Moreover, since it is intrinsically ASCII, XML processing does not depend on any proprietary database management system. The extensible and evolvable data design is also reflected in the metadata design, which adheres to the following principles:

- compliance of content with the OAIS information model to contain information related to unique identifier, provenance, context, fixity, and description
- integration of open XML standards in the archival domain such as PREMIS (PREMIS Editorial Committee 2008) and MIX(Cundiff 2008)
- aggregation of metadata through processing phases
  The processing of an archived digital object can be performed by different archivist groups using varied archival processing systems and technologies. Associated metadata will be collected at each stage, and finally will be accumulated in the metadata. Within this environment, its structure should be designed in such a way to facilitate easy and efficient import of metadata generated by the processing applications. In order to achieve this, the metadata structure can be divided into slots. Each slot is reserved for an archival processing system, and will have disjoint data elements to avoid duplication and complex crosswalk.

All these principles have as a goal for the system to adapt to evolution in the domain and process dimensions. The motivation of our "slotted" structure originates from the fact that different data types need specific metadata elements, besides some common ones such as creator, or creation date. For example, photographs and emails will contain metadata other than textual records. Moreover, our metadata structure design, with its extensible slots, can easily accommodate wholly new data types. For instance, archiving "Interactive Multimedia Performances" would require a new set of metadata to be extracted and created, as suggested by (Ng, et al. 2008).

6.5 Digital Archive Service Composition

The multi-level approach allows WAIS to sustain the evolution characteristics for a normal software system, as well as an archival system.

Indeed, with the SOA paradigm (OASIS 2006), one can argue that WAIS is evolvable to adapt to the three dimensions of system evolution discussed earlier. The diagram shown in Figure 16 depicts the high-level services with standardized data exchange. Since the services such as Ingest, Access, Preservation Planning, and Data Management are defined at a high level, the system will be able to adapt to lower level changes over time. By maintaining the same AIP object for data exchange, stability in the communication between these services and with external systems will be ensured.

The more intriguing question is how WAIS can cope with the evolution relativity inherent to any archival system. We contend that our approach, which is based on SDS and an extensible data structure, offers an efficient solution. While our SDS allows us to define the necessary <<tool>> components, service composition provides service interfaces to the applications and facilitates evolvability within a composition. In the following discussion, we will show the maintainable and

evolvable characteristics of the system with SDS and service composition throughout the phases of the archival process.

6.5.1 Ingest

One important task of the Ingest service is to identify the file formats of ingested materials. In addition to the file format identification, elaborate technical metadata such as original software version should also be extracted and validated. Subsequent operations during Preservation and Access depend heavily on this very first step. By applying SDS to the Ingest service, we can arrive at the design of the FormatIdentify service. Further iterations of the SDS algorithm lead to the specific file format identification tools. One example of a possible data flow can be summarized in the communication diagram shown in Figure 19, where the FormatIdentify service invokes two identification tools DROID and JHOVE in sequence, since the two are complementary as discussed in (Phelps and Wilensky 2001).

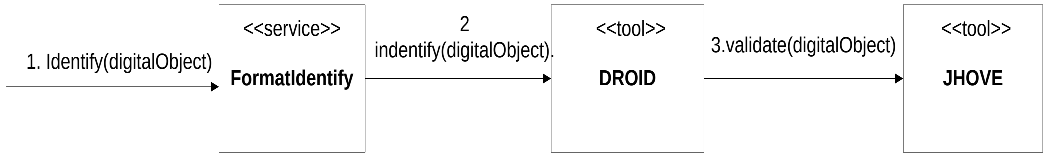

Figure 19 File Identification Process

Now suppose that there is knowledge that new file formats arrive. Then, there are three cases. The easy one is that the tool currently used in the FormatIdentify service only needs to be updated with a new signature as in the case of DROID. A more difficult scenario may require replacing one of the existing tools with a new tool, or just adding a new one to handle this special case. A third case is to augment the current logic flow with a new tool in order to complement the current tool set. In all these cases, if the FormatIdentify service is embodied by a BPEL orchestration, then the maintenance is tantamount to modifying the BPEL script to plug in the new identification tool. Per the SDS presented above, these tools will be available as COTS or FOSS.

6.5.2 Preservation

A similar approach can be used if the preservation strategy of a set of digital objects is Persistent Format Transformation (PFT). PFT is tantamount to running these digital objects through one transformation or a chain of transformations. Thus, if we implement this service as a BPEL orchestration, then we can modify the latter to invoke the appropriate transformation chain, be it a new transformation, or a combination of old and new transformations.

But, in the design, we can have a higher level service called PreservationManager. The communication diagram in Figure 20 shows that PreservationManager invokes one of the three services: Transformation, MultiValentExtractor, or UVCSpecBuilder, depending on whether the strategy specified in the preservation plan is PFT, Multivalent Document, or UVC. The Transformation service will in turn call either PDFConverter or XMLConverter according to the “transform” parameter found in the preservation plan. Note that the UML modeling shows that the interface exposed by PreservationManager remains the same; only its logic or communication steps that decide which preservation strategy to call need to be updated. If the implementation is based on BPEL orchestration, existing strategy can be changed or new strategy added easily. At a lower

level, the system can evolve to add converters other than PDFConverter and XMLConverter to accommodate new preservation formats. Thus, the architecture constructs are reusable, and the interfaces between main services and the invoking applications/services remain intact. Consequently, maintaining and evolving the architecture to cope with new digital object types can be achieved by plugging in new tools into the BPEL orchestration.

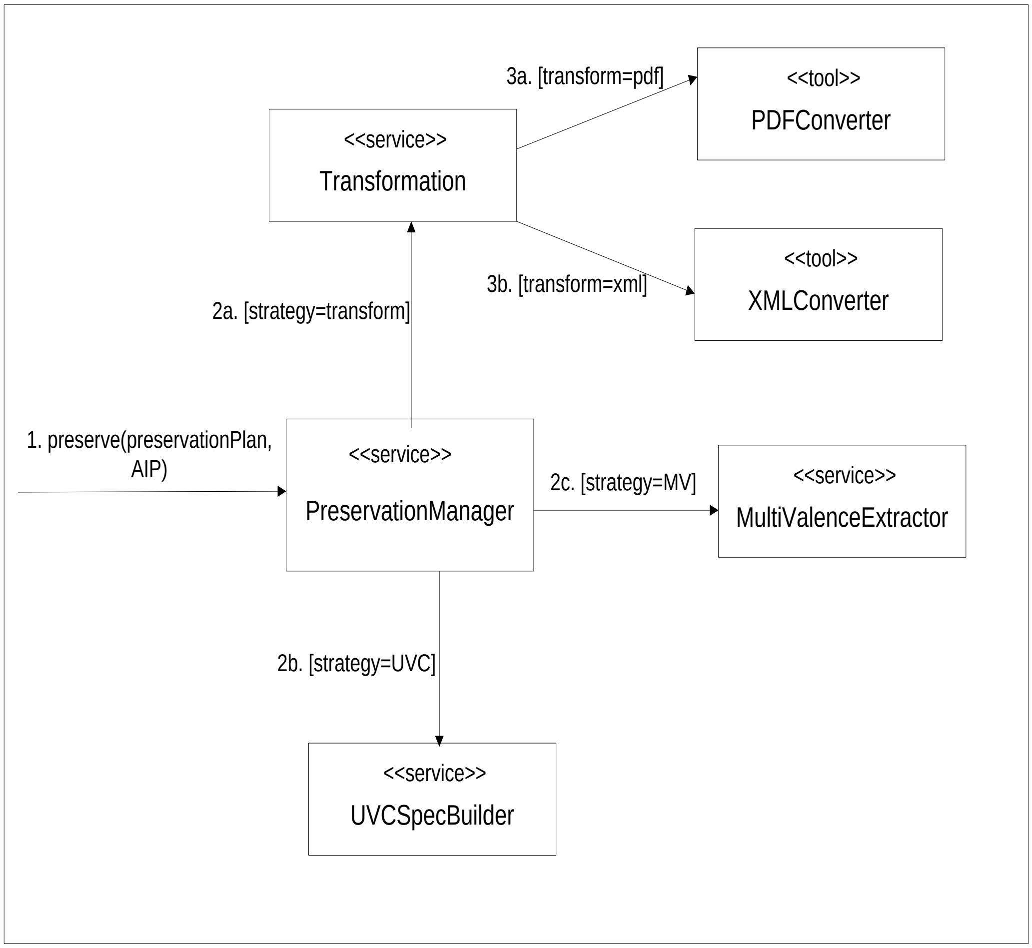

Figure 20 Preservation Manager

6.5.3 Access

The Access component comprises the actions of searching for a digital object, retrieving it, and viewing or "playing" it (as for a multimedia object). Again, we can use BPEL to implement the Search service. The logic for branching is dependent on the types of the digital objects, with each branch leading to a particular search engine. For instance, we can have a search engine dedicated to textual documents, and another for multimedia objects; they will be combined with the metadata search engine as shown by the concurrent tasks 2a and 2b in Figure 21. For viewing or playing, in most cases, the Preservation process will dictate a format that is executable by a standard browser. However, we want to exploit two things from SOA. First, we have a SOA layer dedicated to Presentation as in (A. Arsanjani 2004), so that multi-device and multi-modality can be supported. Second, we envision the participation of the Web 2.0 community to produce innovative tools that can let users manipulate the behavior of a digital object. Another possibility is the use of

a Multivalent Browser (Phelps and Wilensky 2001), which allows the user to perform user-interface operations on the digital objects delivered via the web, such as text highlighting, annotating, and zooming in/out. Such end-user empowerment is only possible if WAIS has extracted, stored, and delivered complete metadata about the digital objects to the Access component. In fact, the flexible FormatIdentify service in the Ingest process, coupled with the metadata structure described earlier in this paper should facilitate, or at least can be improved and evolved to provide a rich set of metadata in the form of a Dissemination Information Package.

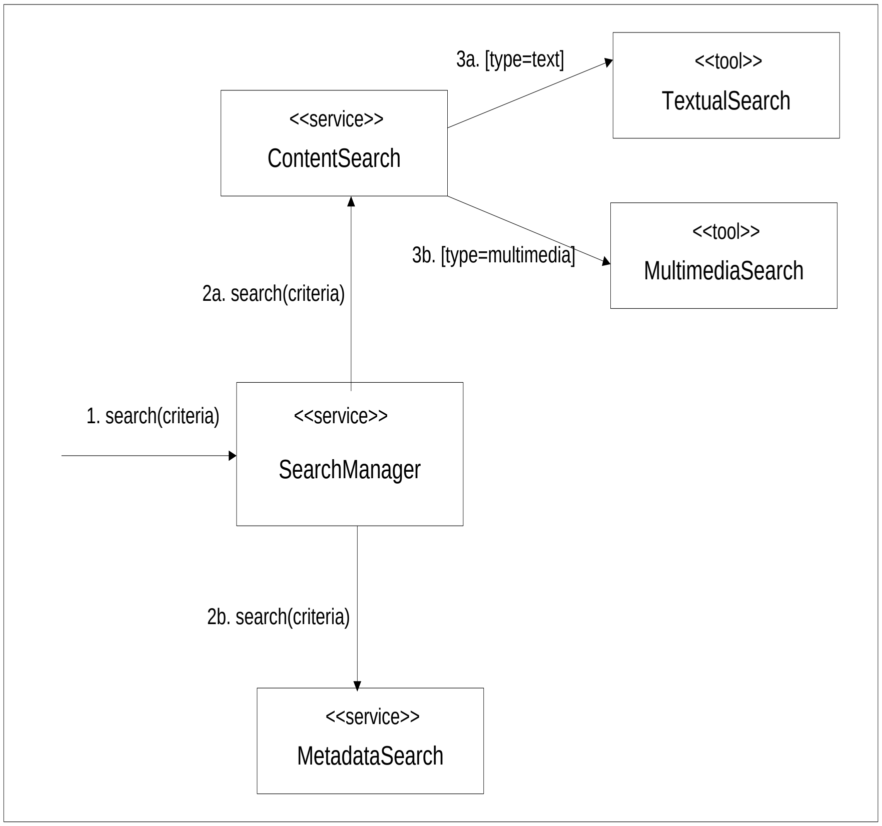

Figure 21 Search Manager

6.6 Conclusions

In this paper, we have presented the issues of system evolution surrounding WAIS. We have shown that WAIS has to cope with two evolution timelines: one for itself, and one for the data that it ingests and preserves for the long term. From this study, we have proposed to: a) enhance the software evolution concept with a data-centric view; and b) facilitate the system evolution of a WAIS and ensure its long life by implementing a multi-level approach. At the architecture level, our approach, which is based on the SOA paradigm, contains a recursive scheme at its core to provide to software architects a practical mechanism to model and design an efficient system evolution and maintenance strategy for a WAIS system. Thanks to this scheme, and to the utilization of current SOA technologies such as BPEL, evolving the system to adapt to new data types or changing data types will be tantamount to monitoring the software market for new preservation-related tools, evaluating them, and plugging them into the framework. Another notable benefit of

the SDS scheme would be to facilitate the specification of new tool-services to be developed in order to keep up with the evolution of digital preservation.

6.7 Acknowledgement and Disclaimer

I would like to thank the anonymous reviewers whose comments have been valuable for the final version of this paper.

The COTS and FOSS components mentioned in this paper are provided for illustration purpose only and cannot be interpreted as the author's endorsement.

Any opinions and conclusions expressed in this paper are those of the author and do not necessarily reflect the views of the U.S. National Archives and Records Administration.

References

Arsanjani, Ali. *Service-oriented modeling and architecture.* November 9, 2004. http://www.ibm.com/developerworks/library/ws-soa-design1/ (accessed February 5, 2010).

Bonardi, Alain, and Jerome Barthelemy. "The Preservation, Emulation, Migration, and Virtualization of Live Electronics for Performing Arts: An Overview of Musical and Technical Issues." *ACM Journal on Computing and Cultural Heritage* 1, no. 1 (2008): 6.

Chaki, Sagar, Andres Diaz-Pace, David Garlan, Arie Gurfinkel, and Ipek Ozkaya. "Towards engineered architecture evolution." *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering.* Vancouver: IEEE Computer Society, 2009. 1-6.

Ciraci, Selim, and Pim van den Broek. "Evolvability as a Quality Attribute of Software Architectures." *The International ERCIM Workshop on Software Evolution 2006.* Lille: University of Twente, 2006.

Consultative Committee for Data Space Systems. *Reference Model for an Open Archival Information System.* Reston, January 2002.

Cundiff, Morgan. *NISO Metadata for Images in XML Schema.* May 12, 2008. http://www.loc.gov/standards/mix/ (accessed February 5, 2010).

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Washington: Addison-Wesley, 1995.

Geeknet, Inc. *DROID (Digital Record Object Identification).* 1999. http://droid.sourceforge.net (accessed February 5, 2010).

JHOVE. *JSTOR/Harvard Object Validation Environment.* February 25, 2009. http://hul.harvard.edu/jhove (accessed February 5, 2010).

National Archives of Australia. *Xena: Digital Preservation Software.* December 9, 2009. http://xena.sourceforge.net (accessed February 5, 2010).

Ng, Kia, Tran Vu Pham, Bee Ong, and Alexander Mikroyannidis. "Preservation of Interactive Multimedia Performances." *International Journal of Metadata, Semantics, and Ontologies* 3, no. 3 (2008): 183-196.

Nguyen, Quyen. "Performance Study of Digital Object Format Identification and Validation Tools." *IEEE Transactions on Power Systems* 11, no. 2 (1996): 851-857.

OASIS. *Reference Model for Service Oriented Architecture 1.0.* October 12, 2006. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html (accessed February 5, 2010).

—. “Web Services Business Process Execution Language, Version 2.0.” *OASIS Standard.* April 11, 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (accessed February 5, 2010).

O'Brien, Liam, Len Bass, and Paulo Merson. “Quality Attributes and Service-Oriented Architectures.” *Proceedings of the International Workshop on Systems Development in SOA Environments.* International Conference on Software Engineering, 2007. 3.

Oracle. *Oracle Outside In Technology.* 2009.

http://www.oracle.com/us/technologies/embedded/025613.htm (accessed February 5, 2010).

Papazoglou, Michael, and Willem-Jan van den Heuvel. “Service-Oriented Design and Development Methodology.” *International Journal of Web Engineering and Technology* 2, no. 4 (2006): 412-442.

Perry, Dewayne E. “Dimensions of Software Evolution.” *Proceedings of the International Conference on Software Maintenance.* Sorrento: IEEE Computer Society, 1994. 296-303.

Phelps, Thomas A., and Robert Wilensky. “The multivalent browser: a platform for new ideas.” *Proceedings of the 2001 ACM Symposium on Document Engineering.* 2001: Association for Computing Machinery, 2001. 58-67.

PREMIS Editorial Committee. “PREMIS Data Dictionary for Preservation Metadata.” *PREMIS Home.* March 2008. http://www.loc.gov/standards/premis (accessed February 5, 2010).

Saul, Christian, and Fanny Klett. “Conceptual Framework for the Use of the Service-Oriented Architecture Approach in the Digital Preservation.” *Proceedings of the Fifth International Conference on Preservation of Digital Objects.* London: The British Library Board, 2008.

Street, Julie, and Hassan Gomaa. “Software Architectural Reuse Issues in Service-Oriented Architectures.” *Proceedings of the 41st Annual Hawaii International Conference on System Sciences.* Waikoloa: IEEE Computer Society, 2008. 316-316.

The InterPARES Project. *The InterPARES Project: International Research on Permanent Authentic Records in Electronic Systems.* http://www.interpares.org (accessed February 5, 2010).

Thibodeau, Kenneth. “Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years.” *Conference Proceedings of the State of Digital Preservation: An International Perspective.* Washington, DC: Council on Library and Information Resources, 2002.

7 SOA Governance Optimizes the Business and Evolution of Service-Oriented Systems

Hausi A. Müller, Priyanka Gupta, Ron Desmarais, Alexey Rudkovskiy, Norha Milena Villegas, Qin Zhu (University of Victoria, Canada), and Leho Nigul (IBM Center for Advanced Studies, Canada)

Abstract

On one hand, SOA governance ensures that the concepts and principles for service orientation and its distributed architecture are managed appropriately and are able to deliver on the stated business goals for services. On the other hand, SOA governance controls the evolution of these service-oriented systems. In this paper, we survey leading governance methodologies designed by different organizations and align them along three critical pillars of SOA governance—people, policies, and processes. We then examine how these governance pillars optimize the business and evolution of SOA systems. At the design-time of a SOA initiative, the people pillar clearly dominates the other two. At run-time, there is a symbiotic relationship among the three SOA governance components. The goals of run-time SOA governance are to control, monitor, and adapt these components and their relationships to optimize business and evolution.

7.1 Introduction

The world of computing is now at a tremendous inflection point. The unprecedented level and speed with which systems, businesses and individuals are able to interact are fueling tremendous innovation across the globe. The ability to integrate and leverage capabilities from a wide spectrum of resources is extremely important, just as the need to align information technology (IT) capabilities with the business for greater speed, agility, and differentiation. Gaining a solid understanding and agreement among stakeholders on how to define, adapt, and maintain business services and processes, as well as to create critical links among them, is challenging. Managing change and evolution across this ever-increasing web of information, services, and systems is truly an uphill battle. With an effective service-oriented architecture (SOA) approach and infrastructure, much common ground can be established, despite the significant variety of SOA components and applications (Rogers 2008)(Rogers 2008).

SOA is an effective paradigm for organizing a system into services that are running in a distributed execution environment and that may be under the control of different ownership domains (IBM 2007). SOA is an architecture paradigm that enables businesses to organize, define, and implement IT projects in such a way that their value can be directly correlated with the business goals and drivers of the enterprise (Brown, et al. 2008). Thus, the primary goal of SOA is to align the business world with the world of IT in a way that makes both domains more effective (High, Kinder and Graham 2005). Moreover, SOA is a way for businesses to model and structure their organizations and operations so that they can efficiently leverage individual capabilities, coordinate their efforts to leverage IT profitably, and reduce complexity. With the proliferation of computing devices and enterprise systems, software systems have evolved from software-intensive systems to systems-of-systems (Software Engineering Institute 2009), and now to ultra-large-scale systems (ULS) and socio-technical ecosystems (Northrop, et al. 2006). While orchestration and

composition of services is currently fairly static, we expect it to become much more dynamic as dynamic service attributes and user contexts become more palatable. Moreover, as environment uncertainty and runtime dynamics become more prevalent in the natural system evolution towards socio-technical ecosystems, flexible and dynamic orchestration will increase in importance.

The distributed nature of services across various business lines promises an enterprise seeking SOA transformation great benefits and, thus, forces the chief architects to make SOA governance the overriding concern in system design. Identifying, specifying, creating and deploying services require SOA governance through a strong and efficient body that can oversee the entire life cycle of an enterprise's service portfolio (Brown and Cantor 2006). Understanding, governing, controlling, and managing environment uncertainty and run-time dynamics of these computing systems—given the onslaught of ever-changing business environment and processes—is a crucial requirement for the survival and success of many businesses (Muller, Kienle and Stege 2009). More than ever, service-oriented systems resort to and rely on self-adaptation and self-management for dynamic service management and composition.

Developers of service-oriented infrastructure have identified people, policies, processes, and technology as key SOA governance pillars (Keen, et al. 2007). These pillars are used to control, manage, and maintain distributed systems of services and resources to optimize business and evolution objectives. In particular, they involve the runtime monitoring of SOA processes to gather key measures such as execution time, availability, throughput, latency, or resource consumption. These pillars also control dynamic service selection, as well as the evolution of services, metadata, and applications in an organization's service-oriented architecture.

SOA governance is implemented by leveraging, extending, and adapting IT governance mechanisms to ensure that the concepts and principles for service orientation and its distributed architecture are aligned sufficiently, managed appropriately, and able to deliver on the stated business objectives (Brown and Cantor 2006). Thus, SOA governance not only affects the quality of business results obtained, but also the effectiveness of the maintenance processes through its deployed SOA infrastructure.

In this paper, we investigate different SOA governance strategies and approaches as published by different SOA providers. We further correlate the approaches and investigate how each strategy can contribute towards an integrated model for SOA governance supporting the optimization of business and evolution objectives. Section 2 discusses three of the pillars of SOA governance—people, processes and policies. Section 3 outlines three different approaches for SOA governance with respect to the three pillars. Section 4 combines these perspectives into a more integrated view and shows how to govern business and evolution objectives uniformly. Section 5 draws some conclusions.

7.2 Pillars of SOA Governance

The most successful SOA initiatives implement a rigorous SOA governance program—the interaction between policies (what), people (who), and processes (how) (cf. Figure 22) to deliver SOA benefits, ensure SOA success, and facilitate evolution (Afshar 2007). It is the oversight of and the interaction between these three components that ensures that SOA provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.

Figure 22 Selected Pillars of SOA Governance (Afshar 2007)

People: The people involved in SOA governance assume several critical roles. SOA governance leaders and strategists establish chains of responsibilities, authority, decision rights, as well as measurement, control, policy, and process mechanisms to enable people to carry out their roles and responsibilities. Responsible and accountable managers and practitioners execute decision rights, measure results, and provide feedback to strategists. Practitioners orchestrate and execute process steps by adhering to, observing, and optimizing specified policies and procedures.

Policies: A hierarchy of policies embodies the decisions that need to be made for effective service management. Policies also include principles and standards that people involved with a SOA governance program create to guide the organization towards the desired behavior and evolution of its SOA solutions (Biske 2008). Policies are created at design-time and adapted at runtime—either manually by people or automatically by processes. Higher level policies define the rules for lower level policy adaptations to optimize business and evolution.

Processes: To orchestrate service-oriented business and to facilitate the evolution of its applications and infrastructure, SOA governance comprises a set of well-defined processes, which enact policies and are executed by people and technology. A process is a sequence of operations or events, possibly taking up time, space, expertise, or other resources, which produces desirable business outcomes according to policies (Brown, et al. 2008). Processes, as well as policies, define multiple levels of the enterprise SOA architecture. Some processes are fine grained and address specific governance needs. Other processes are much more coarse grained and define the governance architecture at the enterprise level.

Thus, services in such an environment are guided by policies, managed by people, and implemented by processes. On the one hand, at design and deployment time the decision-makers (people) are clearly in control of defining chains of responsibility, policies, and processes. Thus, a hierarchical depiction of the relationships between people, policies, and processes would be most appropriate. On the other hand, once the SOA solution is fully deployed, continually delivering value and evolving over time, the balanced, cyclic dependencies among the pillars of SOA governance, as depicted in Figure 22, are more appropriate. In other words, at runtime the three SOA governance pillars are equally important.

7.3 Characterizing the Three Perspectives

Major SOA solution providers emphasize the SOA governance pillars differently. IBM advocates that implementing SOA governance and management requires the consideration of three pillars: people, process, and technology (Mills 2007). Even though the entire governance life cycle is usually described, for the purposes of this paper we will consider the IBM SOA publications that lean towards a process-centric view of SOA governance.

TIBCO's best SOA governance practices paper that concentrates on the critical roles of people in the SOA life cycle is considered for a people-centric view (TIBCO 2005). It is also observed that TIBCO's perspective is mostly relevant for the early design and deployment stages of SOA governance when germinating a SOA initiative.

Oracle purports a more management-centric and policy-centric view. The reviewed Oracle approach is mostly applicable to providing a bridge from the design stage view to the execution view and is particularly useful for validating a SOA governance approach during its installation and deployment from the policy point of view.

All three perspectives constitute useful checklists for the requirements engineering phase of a SOA governance system. Of course, the SOA solution providers concentrate first and foremost on using governance to optimize business objectives. By and large, they do not emphasize that the SOA governance pillars can also be used to optimize evolution objectives.

7.3.1 The IBM Perspective

IBM defines a SOA governance framework as a life cycle consisting of four phases—plan, define, enable, and measure—as depicted in Figure 23 (Holley, Palistrant and Graham 2006, IBM 2009). This framework controls the SOA life cycle, which itself consists of four phases (i.e., model, assemble, deploy, and manage) (IBM 2009).

The activities of the four phases that have to be carried out as part of the SOA governance life cycle are as follows:

Plan: In this phase, understanding the overall scope of governance within the organization is important. Moreover, areas of improvement (i.e., policy and process improvement) are identified in this phase. This phase includes

- defining or refining the overall business vision, strategies, and objectives
- reviewing and revising IT and SOA alignment and capabilities

Define: After identifying the scope and improvements, this phase deals with defining or modifying the governance arrangements and mechanisms including

- establishing the SOA and SOA governance centre of excellence
- defining policies and mechanisms to guarantee service level agreements
- establishing funding mechanisms
- conducting staff training

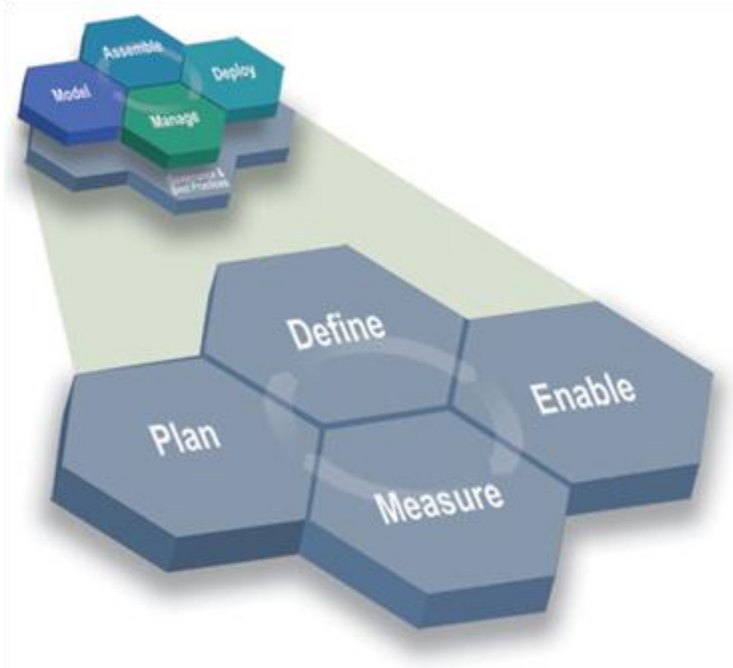

Figure 23 IBM SOA Governance Life Cycle (Holley, Palistrant and Graham 2006)

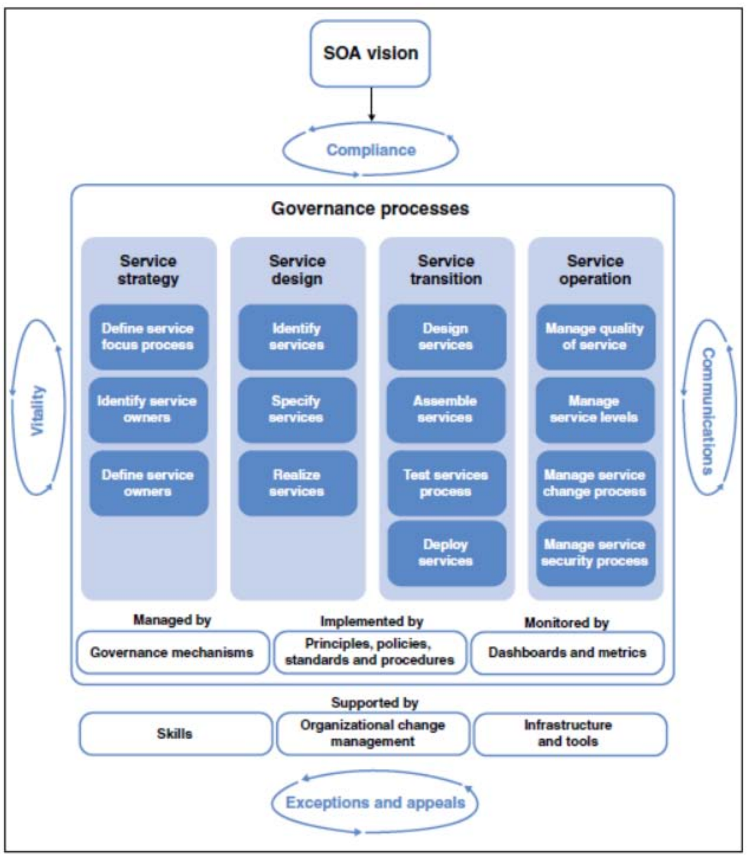

Figure 24 The Key Components of IBM's SGMM (B. Brown 2009)

Enable: All the solutions determined in the above two phases are deployed and put into action. One of the most important observations in this phase is the deployment of SOA with mechanisms to enforce policies. A baseline is established for policy and process improvement. Infrastructure and applications are instrumented to support the Measure phase.

Measure: Governance decisions made in the Define and Enable phases are monitored in this phase. Selected measures of a managed process are fed back to the controller. The controller then tries to optimize business objectives by adapting policies and processes. This enhances the effectiveness of the governance framework and, thereby allows for business tuning and process improvement.

IBM's SOA Governance and Management Method (SGMM) presents a framework for the definition, design and implementation of SOA governance and service life-cycle management (B. Brown 2009). The governance processes of this method are depicted in Figure 24 together with the mechanisms and components that are needed to implement and manage them. The four processes—service strategy, design, transition, and operation—correspond to the four phases of the SOA governance life cycle.

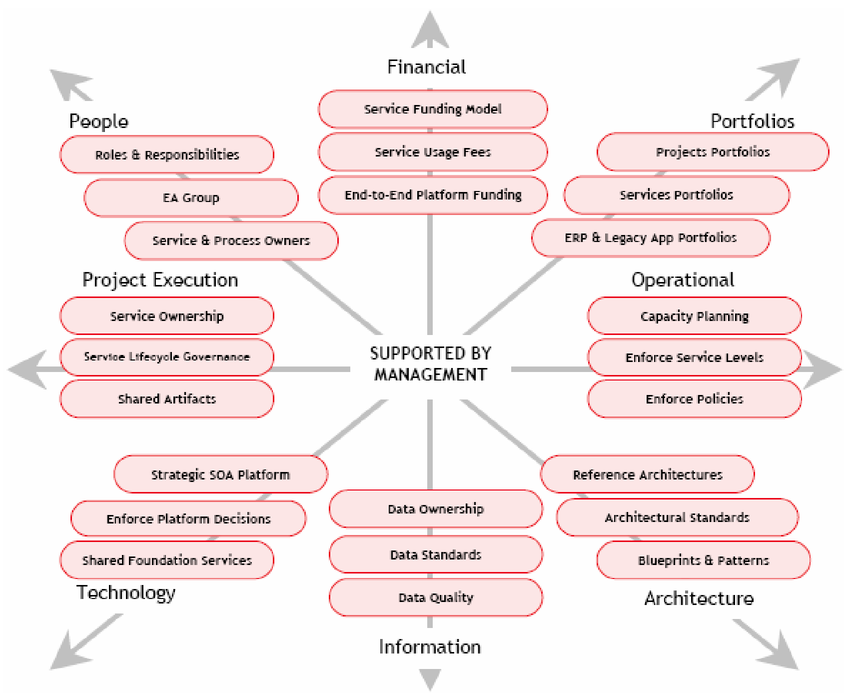

Figure 25 Oracle Leverage Points for SOA Governance Policies (Afshar 2007)

7.3.2 The Oracle Perspective

Oracle's focus is on policies (Afshar 2007). To meet the business goals in the context of enterprise architecture and SOA, policies must be enacted across several dimensions including information, people, architecture, technology, infrastructure, finance, portfolios, projects, and operations (cf. Figure 25). For each dimension, Oracle defines key leverage points to be governed. The policies, which are defined in documents and enforced through technology, define the role of governance to achieve the business and IT goals and alignment between them. To facilitate enterprise-wide SOA adoption, SOA governance experts at Oracle recommend organizing policies and

processes around these leverage points. If all the appropriate policies are put in place, SOA goals can be governed wisely and effectively (Afshar 2007).

7.3.3 The TIBCO Perspective

In addition to processes and policies, people are clearly fundamental to effective SOA governance. TIBCO defines key governance areas from a people perspective, as depicted in Figure 26 (TIBCO 2005). TIBCO defines five different roles and assigns governance capability-based processes to each role.

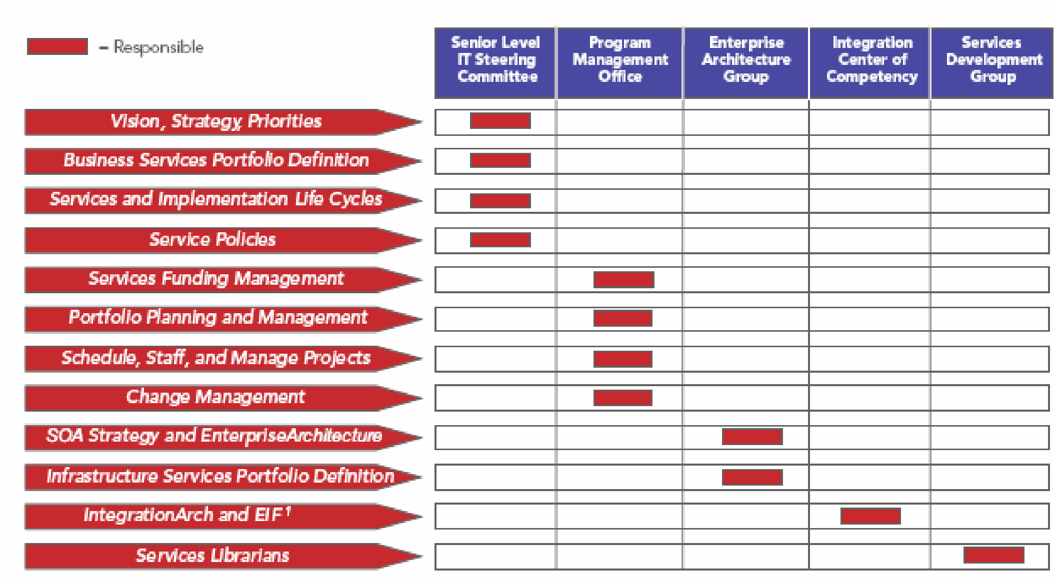

Figure 26 TIBCO Governance People Roles (TIBCO 2005)

The senior level IT steering committee

- defines the vision, strategy and priorities
- ensures that the business portfolio meets the user or customer needs
- designs the services and implementation life cycles
- specifies and governs service policies

The program or project management office

- ensures the funding required for business services
- governs the project and applications portfolio
- schedules, staffs and maintains projects
- handles change management

The enterprise architecture group

- governs and refines the SOA vision, strategy and enterprise architecture
- ensures compliance between the service portfolio and the SOA vision strategy

The integration competency center

- governs the governance architecture
- ensures that it is in tune with related standards, best practices, tools, and reusable components

The services development group

- implements and maintains service repository tools
- enforces policies and standards for service publication
- develops mechanisms to monitor service usage and reuse

7.4 Governing Business and Evolution Objectives

The three pillars discussed in the previous section—people, process, policies—together with the technology pillar constitute different perspectives on the SOA governance life cycle from the early planning stages to the steady-state runtime operation. The entities and relationships introduced also constitute design views for SOA governance methodologies. On one hand, SOA governance ensures that the concepts and principles for service orientation and its distributed architecture are managed appropriately and are able to deliver on the stated business objectives. On the other hand, SOA governance controls the evolution of these service-oriented systems. Naturally, most papers dealing with SOA governance methodologies and best practices concentrate on how to deliver and satisfy business objectives and often do not cover the maintenance and evolution challenges. While not readily apparent, closer inspection reveals that the implementations of such governance methodologies embody classic software evolution concepts, such as levels of indirection and feedback loops.

7.4.1 Governance Feedback Loops

To optimize business functions, service-oriented systems are extensively instrumented to keep track of useful figures such as execution time, availability, throughput, latency, or resource consumption. SOA governance processes use these figures to identify trends, adjust policies and processes, and manage service levels accordingly (Lewis and Smith 2008). In particular, these governance processes monitor and control the effects and outcomes of policies and processes using feedback loops as depicted in Figure 27. Selected results of SOA policies and processes are fed back to a SOA controller, which then decides whether there is a need to adapt policies and/or processes (e.g., variables) to optimize outcomes.

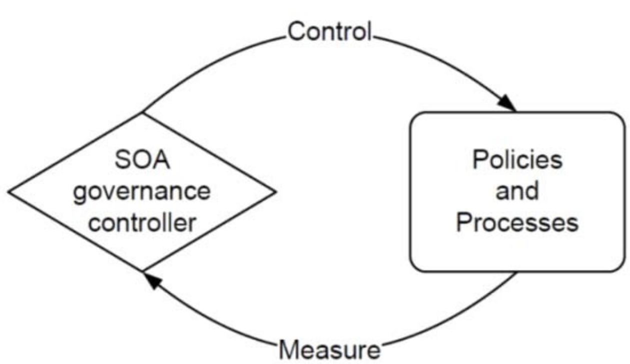

Figure 27 SOA Governance Feedback Loop

Dynamism is both a curse and a blessing. On one hand, it allows for the orchestration of dynamic value sets (Mills 2007). On the other hand, it mandates carefully crafted rules, regulations, and policies for working with these modern, decentralized, distributed socio-technical ecosystems. With the proliferation of dynamic service attributes and user context awareness, replacement of services that are part of a bigger service becomes an important maintenance task. Feedback loops can be used to determine when a better or more appropriate service becomes available by monitoring dynamic service attributes and/or user context variables. For example, a cost variable controller can monitor the dynamic cost attribute of several equivalent services and then adapt the managed process to select the most economical service as depicted in Figure 28.

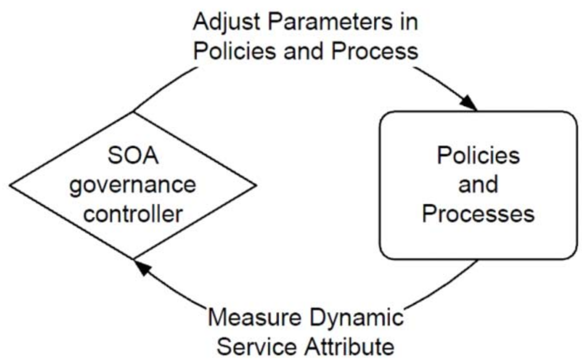

Figure 28 Dynamic Service Attribute Controller

IBM researchers introduced the notion of an autonomic element as a fundamental building block for designing self-adaptive and self-managing systems such as SOA governance controllers (IBM 2005, Mills 2007). An autonomic element consists of an autonomic manager (i.e., controller), a managed element (i.e., process), and two manageability interfaces. The core of an autonomic manager constitutes a feedback loop, referred to as monitor-analyze-plan-execute-knowledge (MAPE-K) loop, as depicted in Figure 28. The manager gathers measurements from the managed element as well as information from the current and past states from various knowledge sources via a service bus and then adjusts the managed element if necessary through a manageability interface (i.e., the sensors and effectors at the bottom of this figure) according to the control objective. Note that an autonomic element itself can be a managed element. The sensors and effectors at the top of the autonomic manager in Figure 28 are used to manage the element (i.e., provide measurements through its sensors and receive control input (e.g., rules or policies) through its effectors). If there are no such effectors, then the rules or policies are hard-wired into the MAPE loop. Even if there are no effectors at the top of the element, the state of the element is typically still exposed through its top sensors.

The autonomic element as an architectural pattern is now well established and has been applied in many different application domains. For example, Brittenham et al. (Brittenham, et al. 2007), (IBM 2005) have distilled common patterns for IT service management (ITSM) based on the best practices in the IT Infrastructure Library (Office of Government Commerce 2006) as depicted in Figure 30.

Similarly, SOA governance management processes can also be characterized using the four autonomic phases (i.e., monitoring, analyzing, planning, and executing). Note that the autonomic architectural pattern applies to both design-time and run-time governance. Often only half of a MAPE loop (i.e., the monitor and analysis phases) is fully automated; the other half is then executed manually. Thus, SOA governance feedback loops can not only be used to optimize its business functions, but also its evolution processes.

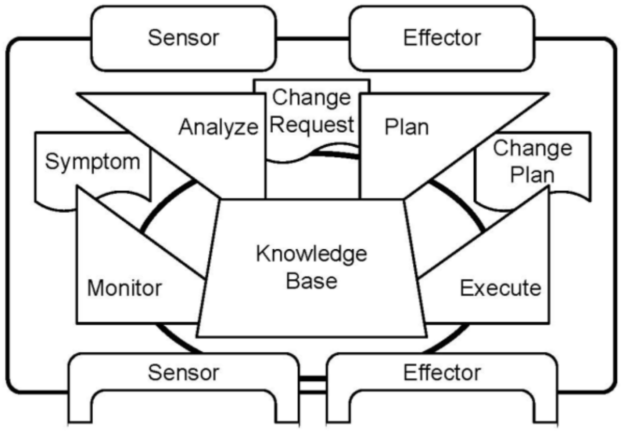

Figure 29 Autonomic Manager

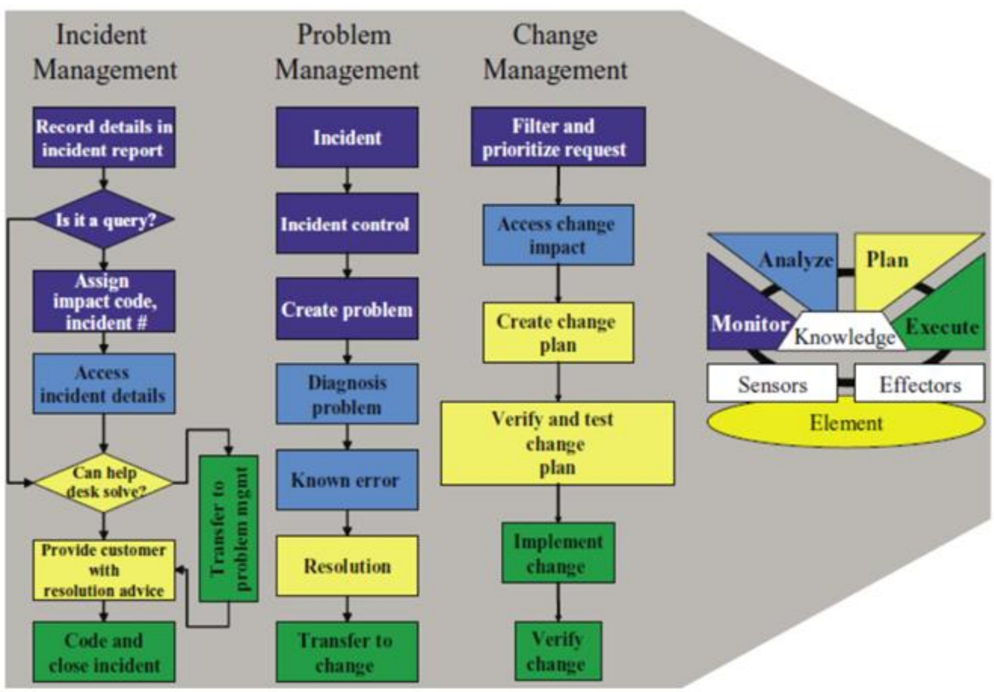

Figure 30 ITSM Processes as MAPE Loops (IBM 2005)

7.4.2 Levels of Indirection

Any problem in computer science can be solved with another layer of indirection.

—David Wheeler

After studying the SOA governance pillars and selected interdependencies, we classified particular entities and relationships according to the time they are most relevant or most critical—design time, deployment time, or steady-state run-time. Design-time SOA governance entities and relationships are often immutable and static, and are thus cumbersome to change and evolve. In contrast, run-time components can be dynamic and are therefore easier to maintain and evolve.

The feedback loops discussed in the previous section are eminently useful for runtime evolution, but do not ease the evolution of static components. However, a classic software engineering strategy—separation of concerns through levels of indirection—can alleviate this problem to a certain extent.

Levels of indirection can be introduced for design time and run-time SOA governance entities and relationships. In particular, levels of indirection can be designed for the key components of IBM's SGMM (outlined in Figure 24) as well as Oracle’s leverage points for SOA governance policies (outlined in Figure 25).

The IBM governance process model with its two-layered governance life cycle—plan, define, enable, and measure—constitutes a level of indirection, which facilitates extendibility and evolution of processes (IBM 2009). Another elegant architectural solution, which is particularly suitable for evolving governance policies, is to organize the policies into layers where the higher level policies orchestrate lower level policies as in the IBM Autonomic Reference Architecture (ACRA) as depicted in Figure 31 (IBM 2005). Policies are fed into autonomic elements through their manageability interfaces. To facilitate extension and contraction of data and information, metaknowledge (e.g., meta schema) is needed. Finally, the people pillar also embodies levels of indirection through chains of responsibilities and decision rights.

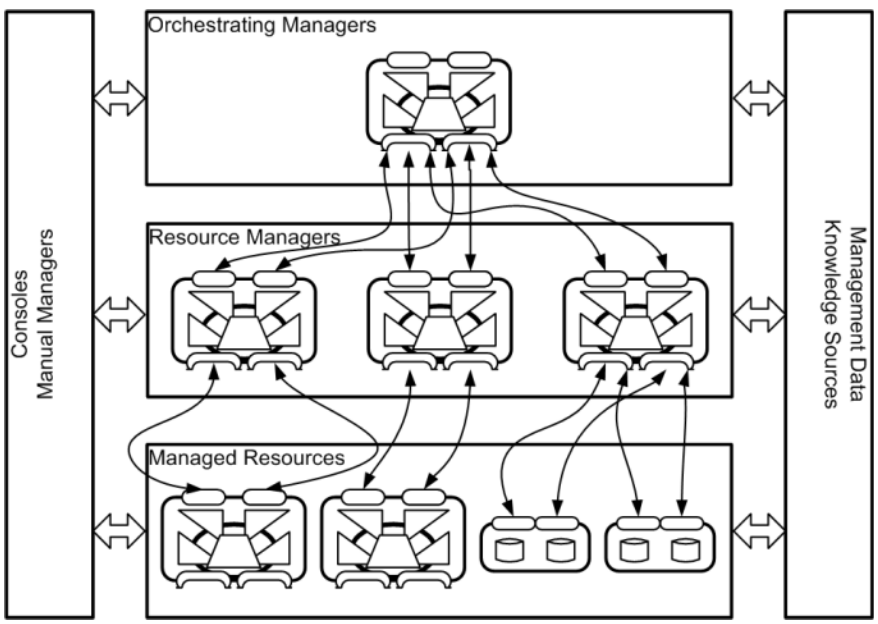

Figure 31 ACRA–Autonomic Reference Architecture

7.5 Conclusions

In this paper, we first defined three important pillars of SOA governance: people, processes and policies. We then identified three methodologies that fit these three dimensions. The three pillars constitute different perspectives on the SOA governance life cycle from the early planning stages to the steady-state runtime operation. We then argued that SOA governance mechanisms cannot only be used to control the delivery of the stated service business objectives, but also the evolution of these service-oriented systems. We then discussed feedback loops and levels of indirection as SOA governance mechanisms.

SOA assures integrity, simplicity, robustness, reuse, ease of maintenance, and agility in a typical business setting. A governance model specifies the processes, polices, controls and governance mechanisms that are required to monitor its service-oriented systems. It also provides the organizational structure and defines the roles and responsibilities that are needed to operate the governance model. Without an effective SOA governance model, it is unclear how the SOA business and evolution objectives can be achieved to the fullest extent.

Levels of indirection and feedback loops are highly effective mechanisms to facilitate and orchestrate maintenance and evolution. These mechanisms are present in the best practices of governance frameworks and service-oriented systems, but might not be as explicit and readily recognized as they should be given their excellent track record and versatility.

References

Afshar, Mohamad. *SOA Governance: Framework and Best Practices.* White Paper, Redwood: Oracle Corporation, 2007.

Bianco, Phillip, Grace Lewis, and Paulo Merson. *Service Level Agreements in Service-Oriented Architecture Environments.* Technical Report, Pittsburgh: Software Engineering Institute, 2008.

Biske, Todd. *SOA Governance.* Birmingham: Packt Publishing, 2008.

Brittenham, P., R. R. Cutlip, C. Draper, B. A. Miller, S. Choudhary, and M. Perazolo. "IT service management architecture and autonomic computing." *IBM Systems Journal* 46, no. 3 (2007): 565.

Brown, Bill. "Introduction to SOA governance and service lifecycle management." *IBM.* March 2009. ftp://ftp.software.ibm.com/software/soa/pdf/IBMSGMMOverview.pdf (accessed February 5, 2010).

Brown, William A, and Murray Cantor. "SOA governance: how to oversee successful implementation through proven best practices and methods." *IBM.* August 2006. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-governance/SOA_governance_how_to_oversee.pdf (accessed February 5, 2010).

Brown, William A., Robert G. Laird, Clive Gee, and Tilak Mitra. *SOA Governance: Achieving and Sustaining Business and IT Agility.* Indianapolis: IBM Press, 2008.

High, Rob, Stephen Kinder, and Steve Graham. "IBM's SOA Foundation: An Architectural Introduction and Overview." *IBM.* December 5, 2005. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf (accessed February 5, 2010).

Holley, Kerrie, Jim Palistrant, and Steve Graham. “Effective SOA Governance.” *IBM.* March 2006. ftp://ftp.software.ibm.com/software/rational/web/whitepapers/soagov-mgmt.pdf (accessed February 5, 2010).

IBM. *An architectural blueprint for autonomic computing.* White Paper, Arnonk: IBM, 2005.

—. *Service Oriented Architecture - SOA.* March 2009. http://www-01.ibm.com/software/solutions/soa/ (accessed February 5, 2010).

—. “Smart SOA: Best practices for agile innovation and optimization.” *IBM.* November 2007. ftp://ftp.software.ibm.com/software/solutions/soa/pdfs/WSW14001-USEN-00_smart_soa_FINAL.pdf (accessed February 5, 2010).

—. *SOA Governance and Service Lifecycle Management.* March 2009. http://www-01.ibm.com/software/solutions/soa/gov/ (accessed February 5, 2010).

Keen, Matrin, et al. *Implementing Technology to Support SOA Governance and Management.* Arnonk: IBM Redbooks, 2007.

Lewis, Grace, and Dennis Smith. “Service-Oriented Architecture and its Implications for Software Maintenance Evolution.” *Frontiers of Software Maintenance.* Beijing: IEEE Computer Society, 2008. 1-10.

Mills, Steve. "The future of business: aligning business and IT to create an enduring impact on industry." *IBM.* June 2007. http://www.ibm.com/services/us/cio/pdf/wp-enbusflex_raw11032-usen-00_hr.pdf (accessed February 5, 2010).

Muller, Hausi A., Holger M. Kienle, and Ulrike Stege. “Autonomic Computing: Now You See It, Now You Don't.” In *Software Engineering: International Summer Schools (ISSSE) 2006-2008, Salerno, Italy: Revised Tutorial Lectures*, by Andrea De Lucia and Filomena Ferrucci, 32-54. Heidelberg: Springer Berlin/Heidelberg, 2009.

Northrop, Linda, et al. *Ultra-Large-Scale Systems: The Software Challenge of the Future.* Pittsburgh: Software Engineering Institute, 2006.

Office of Government Commerce. *ITIL - IT Service Management Books.* 2006. http://www.itil.org.uk (accessed February 5, 2010).

Rogers, Sandra. “Evolving SOA with IBM WebSphere.” *IBM WebSphere.* February 04, 2008. ftp://ftp.software.ibm.com/software/be/solutions/soa/evolving_soa_with_ibm_websphere_02-04-08.pdf (accessed February 5, 2010).

Software Engineering Institute. *Systems Interoperability.* August 31, 2009. http://www.sei.cmu.edu/interoperability (accessed February 5, 2010).

TIBCO. “TIBCO SOA Governance Best Practices: An Introduction.” *TIBCO Software Inc.* 2005. http://www.tibco.com/multimedia/wp-tibco-soa-governance-best-practices-an-introduction_tcm8-2426.pdf (accessed February 5, 2010).

8 Workshop Summary and Next Steps

This section summarizes each workshop session and includes highlights from the discussions that took place among the 20 workshop attendees, who represent both industry and academia. All presentations can be found at http://www.sei.cmu.edu/workshops/mesoa/2009/.

The workshop started with an introduction of the SOA Research Agenda, some of the challenges under maintenance and evolution in the agenda, and a proposed set of maintenance characteristics. The rest of the day was then divided into five sessions:

- Session 1: Tools for Migration to SOA Environments
- Session 2: Case Studies in Systems Migration to SOA Environments
- Session 3: SOA Governance and Service-Oriented Systems Evolution
- Session 4: Longer Term Research Topics in Maintenance and Evolution of Service-Oriented Systems
- Session 5: Panel–Challenges for Maintenance and Evolution of Deployed Service-Oriented Systems

Each session included one or more presentations, plus guided discussion with the goal of identifying remaining research challenges in each area. At the end, the results of the workshop were summarized and next steps were presented.

8.1 Workshop Introduction

Dennis Smith from the Software Engineering Institute (SEI) presented the focus for MESOA 2009, which was to continue sharing current efforts in the maintenance and evolution of service-oriented systems, and identify areas of future work to address existing gaps and provide updates to the research agenda. In addition, the workshop discussed the challenges for maintenance and evolution of service-oriented systems that have already been deployed.

Since this was the third instance of the MESOA workshop, the introduction summarized challenges that were identified in previous workshops, and discussed how these are being followed up. Several of these issues are being actively addressed by the research community, and the research agenda definitely reflects it.

Two major themes from the past two years have been the need for runtime monitoring and self-adaptive SOA systems, as well as effective SOA governance. The current workshop discusses ways to incorporate monitoring within a SOS governance framework. Muller's report in Session 3 incorporates insights from the ongoing ICSE SEAMS workshops, and recommends a SOA governance controller and models for feedback loops and monitoring mechanisms to automate and enforce run-time and change-time governance policies. In Session 4, Kontogiannis provides updated challenges in run-time infrastructure to address specifications for the next generation of SOA infrastructures, and mechanisms to represent SOA governance policies and SOA implementations. Both of these presentations have been influenced by recent advances by Computer Associates and

IBM, and these insights will be factored into the monitoring, SOA governance and maintenance and evolution aspects of the SOA Research Agenda.

The theme of models for ROI in SOA adoption and evolution continues to present a strong challenge because of the inability to generalize across organizations and the lack of explanatory models. The preliminary work with a Carnegie Mellon IT migration that Lewis discusses in Section 2 provides one example of modeling the incremental costs associated with SOA adoption. A summary of ROI challenges is also presented in the overview of the SOA Research Agenda by Grace Lewis.

The need for testing in SOA environments has been cited in past MESOA workshops as a significant research challenge. These challenges include test automation to generate large volumes of test cases, effects of dynamic service composition on testing, rules for service interface definition, and appropriate metrics. These remain significant challenges; testing is discussed in Session 5, the panel session. In addition, a complementary Workshop on Service Oriented Architecture Testing (SOAT) was also conducted at ICSM 2009. Its website is http://www.soatesting.org/2009/.

The current MESOA captures a number of its insights from realistic examples of migration to SOA environments and the evolution of these environments. These include Credit Suisse experience and experiences from Australian organizations reported in Session 5, the case studies of a large telecom organization, and the U.S. National Archives and Records Administration reported in Session 3. In addition the insights reported by Muller in Session 3 and Kontogiannis in Session 4 are based heavily on experiences with Computer Associates and IBM. Collectively these experiences highlight a set of technical issues that include

- low-level communication problems that occur with legacy systems that were never designed to work together, such as different naming conventions
- socio-technical issues, such as upper management support
- the need for standard ways of creating adapters
- the challenge of "double evolution" in which there are different timelines between the evolution of a legacy system and an application developed from services that derive from the legacy
- feedback loops built into runtime governance
- service version management
- SLA management
- service testing
- maintenance and evolution techniques across different levels of abstraction

Grace Lewis from the Software Engineering Institute next presented the SEI SOA Research Agenda, including specific research topics in the maintenance and evolution of service-oriented systems. Most of the discussion focused on whether the maintenance of service-oriented systems was truly different from traditional systems:

- Do we really need completely new maintenance processes?
- Can we augment or adjust existing models?

- Can we leverage maintenance processes from other domains such as manufacturing or networks/telecommunications?

The consensus of the discussion identified the following concerns in the maintenance of service-oriented systems:

- Not all components of the system might reside within the same organization; which makes testing difficult.
- SOA infrastructures tend to be multi-product configurations, each with their own error reporting and handling mechanisms; this makes debugging difficult.
- Unless there is a very elaborate registry and service management mechanism, it is difficult to know exactly who is using what service and in what way.

The discussion also focused on the need for more formal work in return on investment (ROI) for SOA adoption. This is because most of the literature on ROI for SOA is in the form of blog posts or specific vendor experiences. It is difficult to justify SOA investments in the form of ROI primarily for two reasons: a) there is a very large initial investment in infrastructure, and b) some of the benefits of SOA adoption, such as increased business agility, are hard to measure. This would explain why there is a move to measure business value over time, as opposed to calculating ROI. There was also the question of why people are not leveraging the work in engineering economics.

Finally, discussion occurred on how the availability of free services, or services on-demand, would change the model. The agreement was that it changes, both from the consumer as well as from the provider perspective. For the service consumer, it means access to free or on-demand functionality and therefore faster development cycles that can increase business value. For the service provider, it means finding creative business models to increase service consumption and therefore revenue.

The final presentation in the introduction was given by Ned Chapin from InfoSci, Inc. on the maintenance characteristics on SOA systems, based on the paper in Section 3. Its main conclusion is that service-oriented systems are maintenance-intensive systems. Some findings related to successful SOA implementations were:

- Good management is linked to good maintenance.
- SOA projects are mainly maintenance and evolution projects.
- Service-oriented systems are in constant change: system needs and the people that have those needs change. For example, Six Sigma organizations are better prepared for SOA projects because they are used to process improvement and managing change.
- SOA is about much more than technology.

Measures were proposed as potential research projects. These include

- results of SOA projects when managed as maintenance projects
- changes in agility due to SOA adoption
- changes in system size due to SOA adoption
- cost of maintaining deployed SOA systems
- cost of SOA system maintenance by maintenance type
- changes in personnel maintenance time related to SOA adoption

8.2 Session 1: Tools for Migration to SOA Environments

A variety of tools support migration to SOA environments, mainly in web services environments. These include

- modeling tools
- code generation tools for Web Service environments
- complete IDEs for deployment of Web Services
- testing tools

There is also a growing availability of tools and research to support model-based migration to SOA environments—from business models to service models to service implementations that support the business models. Work being conducted in this area was presented by Pooyan Jamshidi from the Shahid Beheshti University GC, based on the paper in Section 4.

The motivation of this work is the accommodation of unanticipated changes in service-oriented solutions in an agile way. The proposed solution is an automated model-based method for evolution of service-oriented systems. Central to the approach is a CRUD (Create, Read, Update, Delete) matrix that indicates what operations are performed by each business process on each business entity. Operations are clustered such that business processes that operate on the same business entities are adjacent. Changes are located in the CRUD matrix and then propagated to the service model through transformations. Research challenges indentified during this presentation and the subsequent discussion were the need for

- common software evolution platforms
- support for model evolution
- support for model co-evolution
- improved predictive models

8.3 Session 2: Case Studies in Systems Migration to SOA Environments

There are a number of case studies about migration to SOA environments. However, it is difficult to draw unbiased, concrete lessons learned, for the following reasons.

- Many of the case studies are sponsored by vendors to showcase tools, methodologies and services.
- Most case studies show successes and not failures.
- Case studies done by academic institutions tend to be examples of “research in the small” with no clear indication whether the particular situation or experiment would scale to an industry setting.

This session had two presentations of unbiased, real case studies. The first was from a Portuguese telecom company, and the second was from the U.S. National Archives and Records Administration.

The first presentation was given by Paulo Rupino da Cunha from the University of Coimbra in Portugal, based on the paper in Section 5. This presentation reported on a telecom organization that experienced three problems in their migration efforts.

- They had to maintain two Business Process Execution Language (BPEL) engines because of long-running processes already running on the old engine. For example, when a new building complex requests telephone lines for their properties, the time between the request and the actual installation can take years. This was called the "recent legacy" problem.
- They found low-level communication problems with their legacy systems because these had never been designed to work together. For example, different systems used different names for the same data element, which caused multiple problems and extra effort to deal with mismatches.
- After services were deployed, the problem was finding services efficiently. The project started experimenting with a semantic registry to address this concern.

Specific research challenges identified during this presentation and discussion included

- Because of incremental evolution, most migration efforts will have to deal with "recent legacy"—their experience was that it was actually easier to deal with old legacy both from a technical and a financial perspective.
- There are multiple socio-technical issues, where governance is just one aspect. Support from upper management is key, as with any migration effort. The use of semantics and ontologies is not just a technical problem, but also a socio-technical issue, because it requires negotiation and coordination between multiple organizations and entities.
- Creating adapters was not as easy as expected. Some services had to expose their internals in order to work. There was not a standard way of calling all applications, and they had problems with rollbacks, especially in long-running processes.
- Creating the link between technology and business is challenging; for example, what is the business cost for run-time service discovery?

The second presentation in this session, based on the paper in Section 6, was given by Grace Lewis of the Software Engineering Institute on behalf of Quyen Nguyen from the U.S. National Archives and Records Administration (who was unable to attend the workshop because of illness).

The presentation discussed the migration of an Electronic Records Archive (ERA) system. The main challenge was to address the "double evolution" problem: dealing with the evolution of the ERA system and with the evolution of the systems that provide the source data to be archived. There are different timelines between the ERA system and the source systems because the gap between creation time and archive time can be years, and the technologies used to create files can be completely obsolete at archive time.

The organization created a service decomposition and composition process to help with evolution. The basic approach involved defining services granularity based on tool availability and taking advantage of BPEL for services composition. This provides easier adaptation of processing and archival processes as technologies evolve.

8.4 Session 3: SOA Governance and Service-Oriented Systems Evolution

IBM defines SOA governance as the process of establishing the chain of responsibilities and communications, policies, measurements, and control mechanisms that allow people to carry out their responsibilities in SOA projects (Woolf 2006). Oracle states that SOA governance provides organizations with the processes, policies, and solutions/technologies that can help to manage increasingly complex SOA deployments in an effective and efficient manner (Oracle 2008).

In general, SOA governance is the set of policies, rules, and enforcement mechanisms for developing, using, and evolving service-oriented systems, and for analysis of the business value of those systems.

There are three types of SOA governance:

- **Design-time governance** applies to early activities such as planning, architecture, design, and development. It includes elements such as rules for strategic identification, reuse, development, and deployment of services, and enforces consistency in use of standards, SOA infrastructure, reference architectures and processes.
- **Run-time governance** applies to deployment and management of service-oriented systems. It includes policies and enforcement mechanisms to ensure that services are executed according to policy and that important run-time data are logged.
- **Change-time governance** applies to maintenance and evolution of service-oriented systems. It includes policies and enforcement mechanisms for maintenance and evolution, as well as communication of changes to stakeholders.

Of particular interest to this workshop is change-time SOA governance, which includes policies that address the following questions:

- How are service changes and upgrades decided and communicated?
- Who pays for maintenance and development of shared services?
- How do governance processes support rapid re-verification of functional capability and system qualities in the event of a new version?
- What happens when a service changes?
- What types of changes lead to complete revalidation? What changes do not?

Hausi Müller from the University of Victoria in Canada gave a presentation on SOA governance and its relationship to the business and evolution of service-oriented systems, based on the paper in Section 7.

The main challenge identified in this presentation is that most SOA governance methodologies do not cover maintenance and evolution challenges systematically. However, the fact that service-oriented systems—like control systems—have feedback loops can be exploited to help in the automation and enforcement of change-time governance.

In this model, policies and processes are continuously measured and controlled by a SOA governance controller to validate existence, compliance, etc. Specific SOA governance mechanisms include autonomic systems and concepts such as feedback loops and levels of indirection.

Research challenges include

- definition of models for feedback loops
- mechanisms for managing and leveraging uncertainty
- mechanisms for making control loops explicit in service-oriented systems

Questions that came up during the discussion included whether maintenance for service-oriented systems was easier than for traditional systems, and whether software co-evolution was harder or easier.

8.5 Session 4: Longer Term Research Topics in Maintenance and Evolution of Service-Oriented Systems

Kostas Kontogiannis from the National Technical University of Athens gave a presentation of longer term research topics in the maintenance and evolution of service-oriented systems. The goal for this presentation was to show a perspective on how service-oriented systems are evolving and what maintenance and evolution challenges this is creating now and for the future.

In the end, the targets for service-oriented systems development and deployment are

- simplified development of business services
- simplified assembly and deployment of business solutions built as networks of services
- increased agility and flexibility
- protection of business logic assets by shielding from low-level technology changes
- improved testability

What follows is a set of research topics mentioned during the presentation that would help to better reach these goals. More details can be found in the presentation on the workshop website.

SOA Programming Models. A SOA programming model can be defined as a collection of models, techniques, methodologies and tools for implementing services and assembling them into solutions. Examples of SOA programming models include IBM's Service Component Architecture (SCA) (IBM 2006), Microsoft's Indigo (Microsoft Corporation 2005), and Sun's Java Business Integration (JBI) (Sun Microsystems 2010). The main goal of SOA programming models is to be able to express business logic at a high level and via model transformation generate code that implements services that express the business logic, as well as the infrastructure to manage these services. The major issues in this area are related to simplification and standardization:

- common definition of a service
- simplified data access models
- component types to simplify development
- better ways to represent process components
- customization of services via design patterns, templates, and points of variability
- development tools for the full service-oriented system life cycle

System Management and Runtime Infrastructure. After service-oriented systems are deployed they have to be managed to ensure that system goals as well as business goals for SOA adoption are being met. This requires a runtime infrastructure that can be instrumented to capture pre-defined measures, service-level parameters and simply guarantee that the system is functioning according to requirements. Topics of research interest in this area include

- root cause analysis to determine sources of error in multi-product SOA infrastructures that provide access to multi-platform, heterogeneous services
- business process view of logs that provide a higher level and consolidated abstraction of heterogeneous logs
- mechanisms to represent SOA governance policies and ensure compliance with these policies
- assurance and security, especially in multi-organizational SOA implementations
- specifications for the next generation of SOA runtime infrastructures

SOA Maintenance and Evolution. As mentioned earlier, as service-oriented systems are deployed, their maintenance and evolution is a concern. In this area, topics of interest include

- evolution patterns of service-oriented systems
- round-trip engineering from business models to services models and from service models to business models
- reengineering processes to support migration to SOA environments
- tools and environments to support maintenance activities in SOA environments
- multi-language and multi-platform system analysis and maintenance
- tools for the verification and validation of compliance with design and runtime constraints

SOA Requirements Engineering. Requirements activities in SOA environments focus on business process modeling, service reuse and change management. To improve requirements engineering in this environment, additional research in this area would be of benefit. This research may include

- modeling dynamic configurations of service-oriented systems that may not be fully composed until runtime
- modeling and managing conflicting non-functional requirements from different system stakeholders
- better understanding of design decisions and tradeoffs for non-functional requirements
- risk assessment and mediation for service sourcing options
- infrastructures for change control and management

Service Versioning. Service versioning so far has focused on techniques for maintaining different versions of a same service without disrupting service consumers. Research in this area should expand this concept to account for the complexity and diversity of services as well as service infrastructures. Research in this area may include further work on

- models and techniques that allow for the automatic or semi-automatic identification of mismatches or potential mismatches between the service as specified and the service as deployed, beyond pure syntactic differences to focus on behavioral and operational aspects as well
- techniques that allow for global service updates including models and tools for what-if analysis during component substitution
- programming model extensions for specifying, denoting and validating whether data sources and schemas are complete and adequate with respect to the data required for a composition of deployed services
- abstractions, models and techniques to manage service connectivity (e.g., wiring properties) throughout the life cycle, without the need of altering the overall service assembly and supporting infrastructure

Property Tracing. Impact analysis was mentioned earlier as an important activity in maintenance and evolution of service-oriented systems in order to determine the impact of system change on service consumers. An extension of this work could include risk analysis and focus on overall system behavior and system properties from design-time to runtime. Research topics for this area may include

- techniques to allow for the acquisition of data that can assist analysts and developers determine how quickly a service can be provisioned to accept new consumers and implement with minimal impact the required adaptations
- techniques and models that allow for the acquisition of data that can assist analysts and developers determine the impact of a change in a service or wiring between services to other services and to the system behavior in general
- techniques and models to assess the vulnerability of a SOA system with emphasis on security issues, as well as techniques and models to assess the risk and impact of service unavailability

Smart Service Infrastructures. The focus of this research topic is to move to the infrastructure tasks that are currently performed by developers or that have to be added to the infrastructure as a non-standard feature. Examples of these tasks include

- techniques for dynamic identification of boundaries and scope of transactions with the goal of maintaining integrity or recovering/compensating from the inability to successfully perform a service or set of services
- techniques for context-aware semantic and operational type service descriptions to increase the effectiveness of service discovery by service consumers looking for service with specific characteristics, design-time properties or runtime properties

- techniques, models, formalisms and framework extensions to allow for different conversation policies

Logging, Monitoring and Diagnostics in SOA Environments. The focus of this research topic is to take logging, monitoring and diagnostics to a new level that not only covers all elements of service-oriented systems, but also helps with early detection and reaction to failures. Research areas in this topic may include

- techniques for the non-intrusive collection of events in a customized or adaptive manner that facilitates the processing and analysis of collected events efficiently
- techniques for the acquisition of data that can assist analysts and developers to perform root cause analysis in large loosely coupled SOA systems
- techniques to allow for "pre-flight" and "in-flight" checks
- techniques for seamless and continuous crosscutting monitoring of SOA systems at the infrastructure, application and business process abstraction layers

Data-Centric Services and Infrastructure. Many service-oriented implementations are data-centric, especially those that access, handle and validate data that exists in legacy systems. For these types of implementations, research in this area is important. Research topics in this area may include

- techniques and infrastructure to attain and maintain global awareness of all data changes across services and systems, regardless of the source of change
- techniques and programming model abstractions for efficient caching and distribution of data into a distributed data/cloud type of architecture

8.6 Session 5: Panel—Challenges for Maintenance and Evolution of Deployed Service-Oriented Systems

The panelists in this session were asked to present what they believed were the greatest challenges for the maintenance and evolution of service-oriented systems. The invited panelists were

- Kostas Kontogiannis (National Technical University of Athens, Greece)
- Hausi Müller (University of Victoria, Canada)
- Liam O'Brien (National ICT, Australia)
- Scott Tilley (Florida Institute of Technology, USA)
- Carl Worms (Credit Suisse, Switzerland)

Liam O'Brien from National ICT in Australia could not be present at the panel, but sent in his nominations for the top challenges in maintenance and evolution of service-oriented systems. His thoughts come from direct experience with organizations in Australia that he is working with in their SOA development efforts.

- Service Version Management. *"An organization that I was involved with had about 12–15 deployed services and they said that they could at any time release and maintain up to 10*

versions of each of those services. This becomes a nightmare with a possibility of 120–150 services available."

- Service Testing. *"Another challenge is testing of services, especially doing performance and scalability assessments. How do you do this in production environments when others users could be using the services?"*
- SLA Management. *"If one changes services what impact will that have on SLAs for that service? Does one have to notify all users of changes in SLAs? If, for example, one added additional functionality to a service and the response time of a service is increased from four seconds to five seconds what will the impact be on users of that service? What if certain users really wanted the four-second response time?"*

The next panelist was Carl Worms from Credit Suisse. He has managed one of the largest SOA implementations in the world. In the financial industry, 20 to 25 percent of employees are in the IT department. This is understandable given how critical information systems are to business operations and revenue. From his perspective, the greatest challenges are:

- At Credit Suisse, and at many financial organizations, services are mostly mainframe application encapsulations. As a result, mainframes and applications still need to be maintained, in addition to the new services.
- Given the size of IT departments, and application and platform diversity, governance is critical. The key is to keep it simple enough, such that goals are met but also so that it does not become a burden for the IT department and developers.
- The identification of small chunks that can be managed and maintained easier is extremely important. However, this can be difficult, especially when there is strong coupling between systems and applications.
- Getting people with the right skills is a known problem—the mix of technical as well as business skills is not easy to find.
- The biggest challenge is maintaining parallel versions of a service in production. To deal with the problem, in his organization each service is restricted to three parallel versions, where the oldest has to be phased out in a year.

Scott Tilley, from the Florida Institute of Technology, separated challenges into the four areas of the SEI SOA Research framework. His perspective is from his work and interests in SOA testing and SOA education.

- **Engineering**. In this area the main challenge is managing the complexity of deployed systems. Engineers need better techniques for SOA system documentation and understanding. An additional challenge given the complexity is SOA testing—the ability to test all aspects of a SOA system.
- **Business**. There is a need for better models for ROI for SOA adoption that takes into consideration system needs after deployment. As mentioned earlier, evolution patterns for SOA systems would provide insight into how these types of systems evolve. From an academic perspective, there needs to be more case studies.

- **Operations**. Longitudinal case studies that cover several versions or releases of a SOA project would provide better insight into what happens over the life of a system. Also, given the increasing adoption of cloud computing it is important to understand how SOA-based systems fair in this broader context.
- **Cross-Cutting**. The challenges in this area are no different than for new systems: SOA governance, security and education.

Kostas Kontogiannis, from the National Technical University of Athens, mentioned the following points as the main challenges in the maintenance and evolution of service-oriented systems. His perspective comes from academia as well as multiple joint projects with IBM and Computer Associates related to systems modeling and model transformation in service-oriented environments.

- Having maintenance and evolution techniques across different levels of abstraction—application, infrastructure and business processes—can help parts of the system evolve as needed.
- Lack of skilled developers is a problem. Developers need knowledge of technologies and platforms that are moving targets. Proper abstractions at different levels could be a solution to this problem.
- There needs to be better dependency models and dependency analysis techniques to analyze the impact of system changes (e.g., changes between applications and infrastructure components).
- Multi-language and multi-platform system analysis was an area mentioned earlier that could help to deal with the heterogeneity that is common in service-oriented systems.
- Service-oriented systems contain components that are not source code (e.g., business process descriptions, service descriptions, configuration files). Guidance for maintenance and evolution of components other than source code would be of great benefit.
- Given the growing interest in cloud computing, it is important to understand the effects of cloud computing on service-oriented systems
- It would be of great benefit to characterize and understand maintenance and evolution challenges in particular industry domains.

Finally, Hausi Müller from the University of Victoria presented his challenges in the maintenance and evolution of deployed service-oriented systems. His perspective comes from his work and interest in using autonomic computing for monitoring and adaptation of service-oriented systems, as well as his work with IBM and Computer Associates.

- Automated root cause analysis—it implies taking traceability to a new level to include dynamic aspects.
- Governance—it is necessary to bring out the importance of SOA governance in the full service-oriented system life cycle, from planning all the way to maintenance and evolution.
- Dynamic attributes—use techniques to deal with system attributes that are not known until runtime, or that change at runtime.
- Context-awareness—service-oriented systems must take user context and execution context into account for dynamic system configuration and service binding

The panel presentations had several challenges in common

- service versioning—dealing with multiple deployed versions of services
- training and education—in all aspects of service-oriented systems
- testing—especially end-to-end, multi-layer testing
- integration and leverage of cloud computing—although there were mixed opinions about whether this should be part of an already broad research agenda
- root cause analysis—for troubleshooting in complex, multi-layered, multi-component service-oriented environments

Two topics emerged from the panel discussion as areas that would make it easier to maintain and evolve service-oriented systems.

- Governance-driven development. SOA Governance is not a new topic. However, the problems with many of the current SOA models are that they are very technology-centric, too comprehensive, and focus mainly on the runtime aspects of service-oriented systems. There are unique challenges and opportunities to do it right, both at design-time and runtime. For example, from a design and engineering perspective, SOA governance is the way to enforce system architecture. Also, not all organizations need the same amount of governance. There needs to be more work between industry and academia to see how service-oriented systems are implemented and what governance is really needed throughout the life cycle. An open question then becomes what is the threshold is for determining how much governance is enough.
- Lean SOA. As organizations start relying on SOA for increased agility, systems have to be able to respond to this request for agility. It would be interesting to leverage lean concepts and practices from industrial engineering such that systems are clear, concise, to the point, and less expensive. This would include the investigation of technologies that support lean development (e.g., REST). It would also include guidance and practices to place controls where they are really needed.

8.7 Next Steps

The workshop ended with a summary and a set of next steps that included the publication of these proceedings and a commitment to continue updating the research agenda to reflect advances in the field and new challenges that occur in the maintenance and evolution of service-oriented systems.

The next steps for this project are the publication of the full SOA Research Agenda.

In 2010, there will be two events related to the evolution of the SEI SOA Research Agenda:

- 2nd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS 2010) in conjunction with the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010) on May 1-2, 2010 in Cape Town, South Africa (http://www.s-cube-network.eu/PESOS).

- 4th International Workshop on Maintenance and Evolution of Service-Oriented Systems (MESOA 2010) in conjunction with the 26th IEEE International Conference on Software Maintenance (ICSM 2010) on September 17, 2010 in Timisoara, Romania.

References

IBM. *Service Component Architecture.* 2006. http://www.ibm.com/developerworks/library/specification/ws-sca/ (accessed February 5, 2010).

Microsoft Corporation. *Introducing Indigo: An Early Look.* 2005. http://msdn.microsoft.com/en-us/library/aa480188.aspx (accessed February 5, 2010).

Oracle. "Increasing the Effectiveness and Efficiency of SOA Through Governance - 2008 SOA Governance Survey Report." *EbizQ.* 2008. http://www.ebizq.net/white_papers/10075.html (accessed February 5, 2010).

Sun Microsystems. "Java Business Integration." In *Sun Java System Application Server 9.1 Administration Guide*, by Sun Microsystems, 49-52. Santa Clara: Sun Microsystems, Inc., 2010.

Woolf, Bobby. *Introduction to SOA governance.* June 13, 2006. http://www.ibm.com/developerworks/library/ar-servgov (accessed February 5, 2010).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	February 2010	Final

4. **TITLE AND SUBTITLE**
Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009)

5. **FUNDING NUMBERS**
FA8721-05-C-0003

6. **AUTHOR(S)**
Grace Lewis, Dennis B. Smith, Ned Chapin, & Kostas Kontogiannis

7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. **PERFORMING ORGANIZATION REPORT NUMBER**
CMU/SEI-2010-SR-004

9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. **SUPPLEMENTARY NOTES**

12A **DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified/Unlimited, DTIC, NTIS

12B **DISTRIBUTION CODE**

13. **ABSTRACT (MAXIMUM 200 WORDS)**

The 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009) was held at the 25th International Conference on Software Maintenance (ICSM 2009) in Edmonton, British Columbia, Canada on September 21, 2009.

The goal for MESOA 2009 was to discuss current efforts in the maintenance and evolution of service-oriented systems and identify research challenges in addressing existing gaps and problems. The workshop had 20 attendees, representing industry and academia. Papers and presentations contributed original work, and the discussions highlighted additional work that is being done as well as new research challenges. Presentations from the workshop can be found at http://www.sei.cmu.edu/workshops/mesoa/2009/.

This report includes selected papers presented at the workshop that highlight important issues within the established SOA Research Agenda.

14. **SUBJECT TERMS**
SOA, service-oriented architecture, service-oriented systems, maintenance, evolution, legacy systems, migration, governance, design

15. **NUMBER OF PAGES**
107

16. **PRICE CODE**

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL